

Spring MVC Security Overview



You will learn how to ...

- Secure Spring MVC Web Apps
- Develop login pages (default and custom)
- Define users and roles with simple authentication
- Protect URLs based on role
- Hide/show content based on role
- Store users, passwords and roles in DB (plain-text -> encrypted)

Practical Results

- Cover the most common Spring Security tasks that you will need on daily projects
- Not an A to Z reference ... for that you can see **Spring Security Reference Manual**

<http://www.luv2code.com/spring-security-reference-manual>

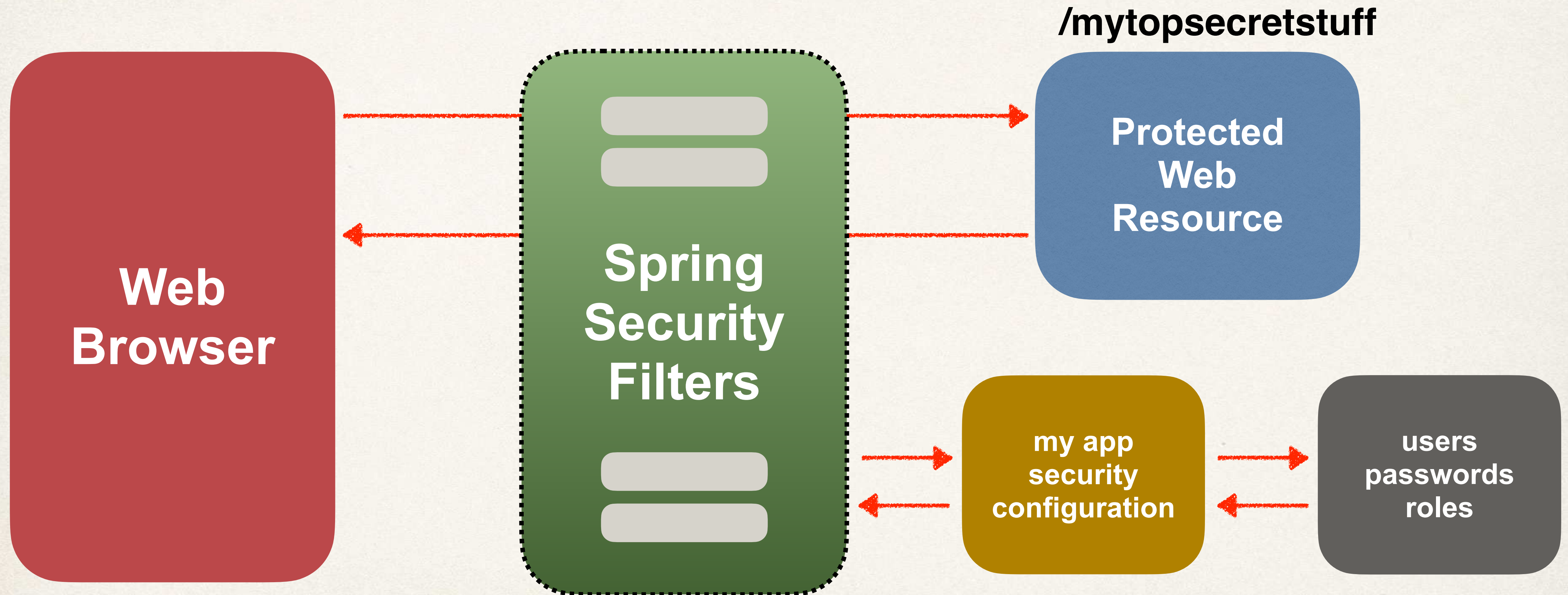
Spring Security Model

- Spring Security defines a framework for security
- Implemented using Servlet filters in the background
- Two methods of securing an app: declarative and programmatic

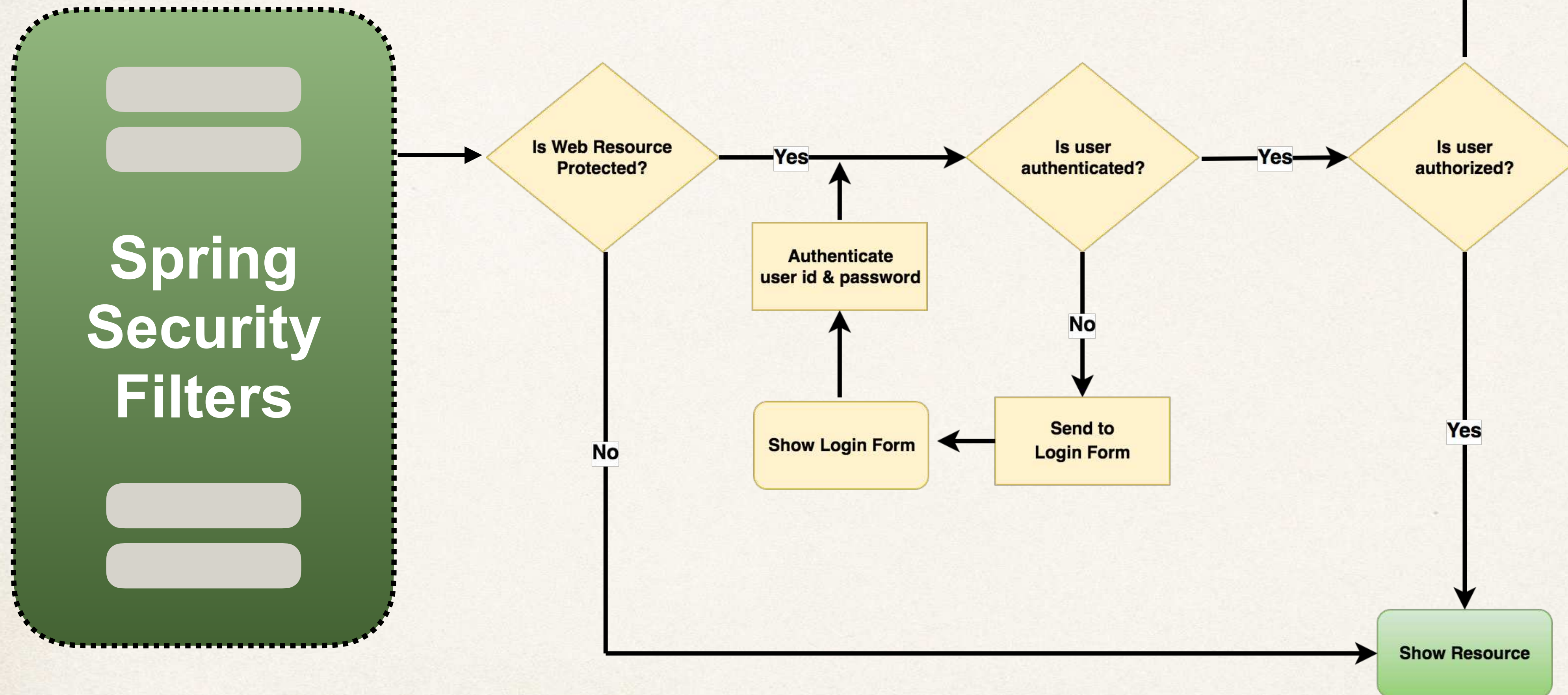
Spring Security with Servlet Filters

- Servlet Filters are used to pre-process / post-process web requests
- Servlet Filters can route web requests based on security logic
- Spring provides a bulk of security functionality with servlet filters

Spring Security Overview

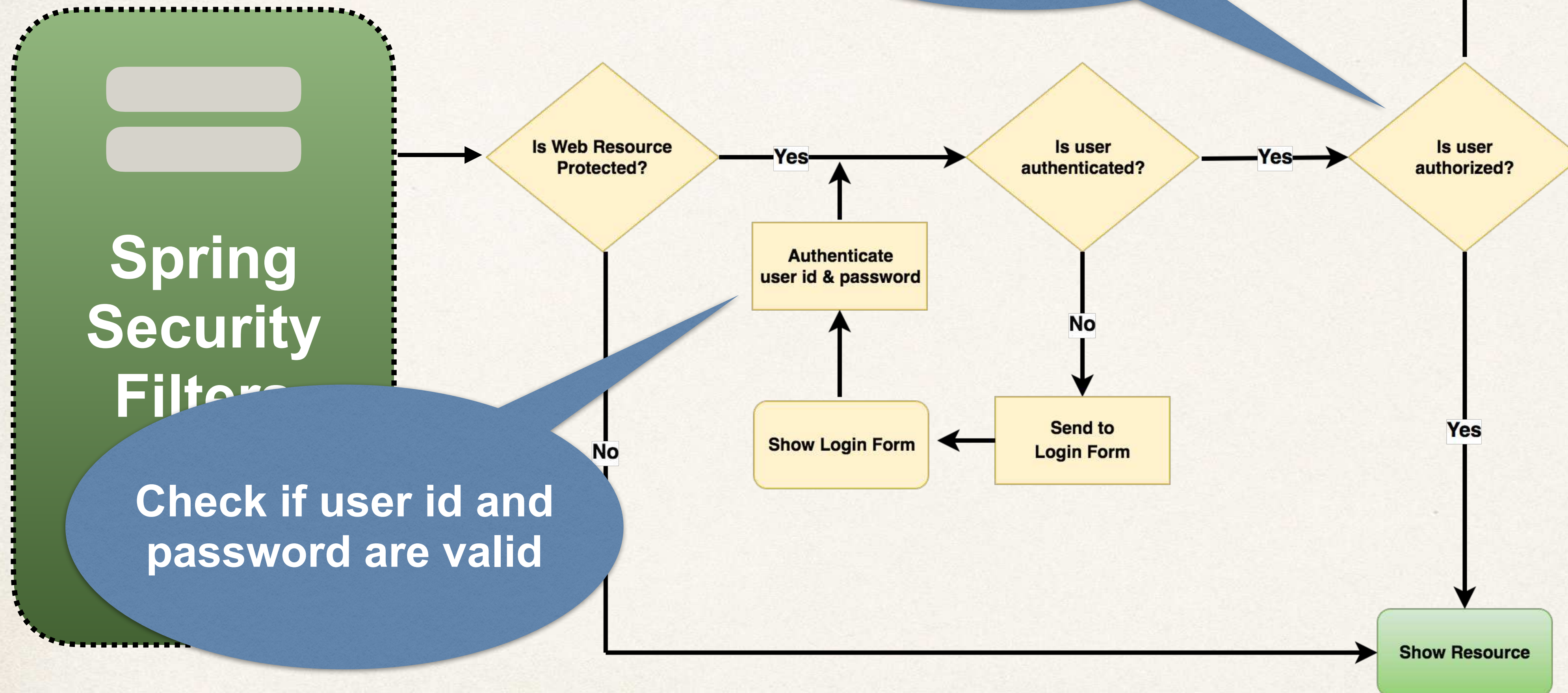


Spring Security in Action



Spring Security in Action

Does user have authorized role?



Check if user id and password are valid

Security Concepts

- Authentication
 - Check user id and password with credentials stored in app / db
- Authorization
 - Check to see if user has an authorized role

Declarative Security

- Define application's security constraints in configuration
 - All Java config: @Configuration
- Provides separation of concerns between application code and security

Programmatic Security

- Spring Security provides an API for custom application coding
- Provides greater customization for specific app requirements

Enabling Spring Security

1. Edit `pom.xml` and add `spring-boot-starter-security`

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-security</artifactId>  
</dependency>
```

2. This will *automagically* secure all endpoints for application

Secured Endpoints

- Now when you access your application
- Spring Security will prompt for login

Please sign in

Username

Password

Sign in

Default user name: **user**

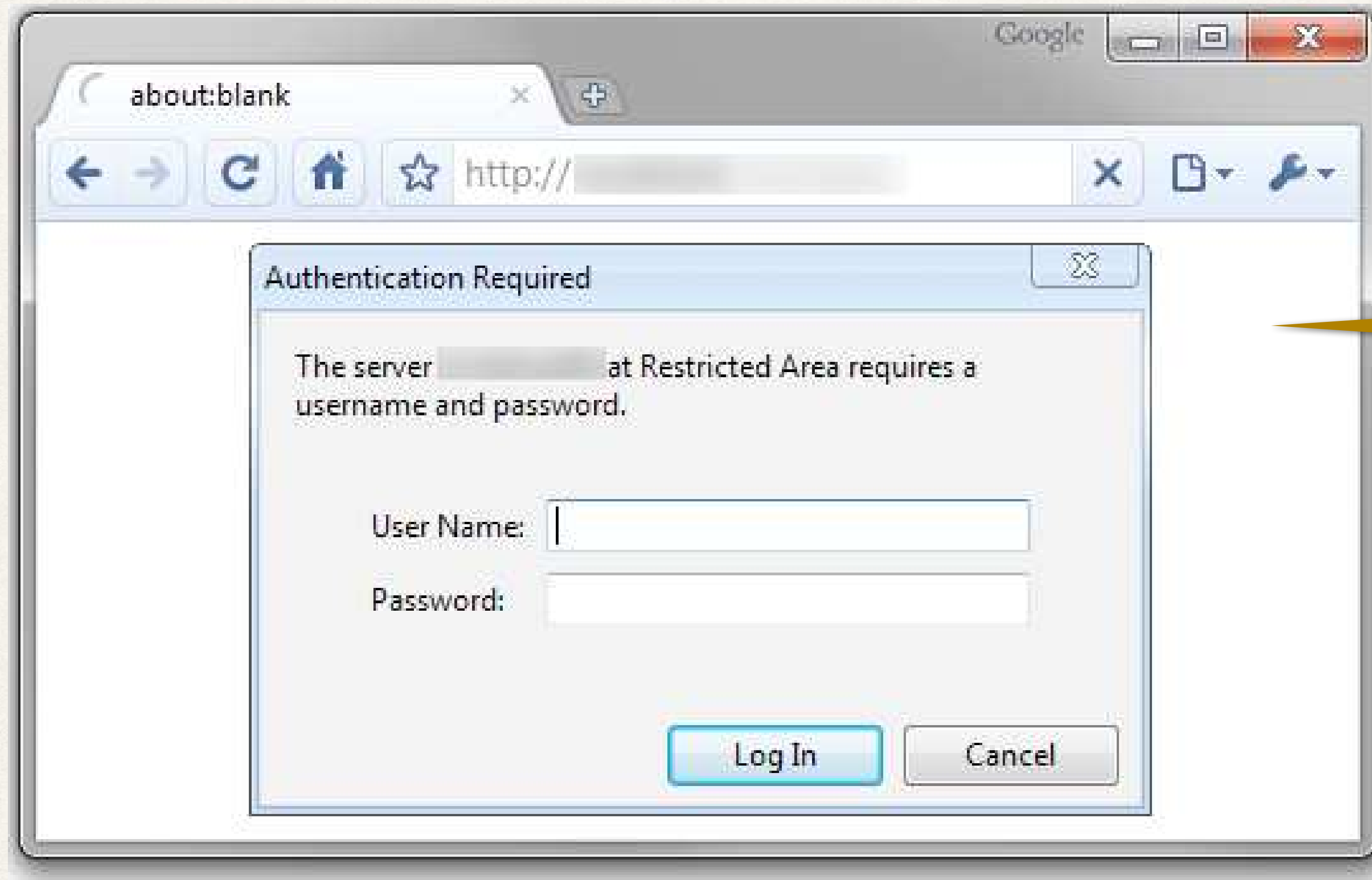
Check console logs
for password

```
11-02 21:05:57.074 INFO 24986 --- [main] .s.s.UserDetailsSe  
Using generated security password: 78fd68a6-c190-421d-934b-df7852fc7dc2
```


Different Login Methods

- HTTP Basic Authentication
- Default login form
 - Spring Security provides a default login form
- Custom login form
 - your own look-and-feel, HTML + CSS

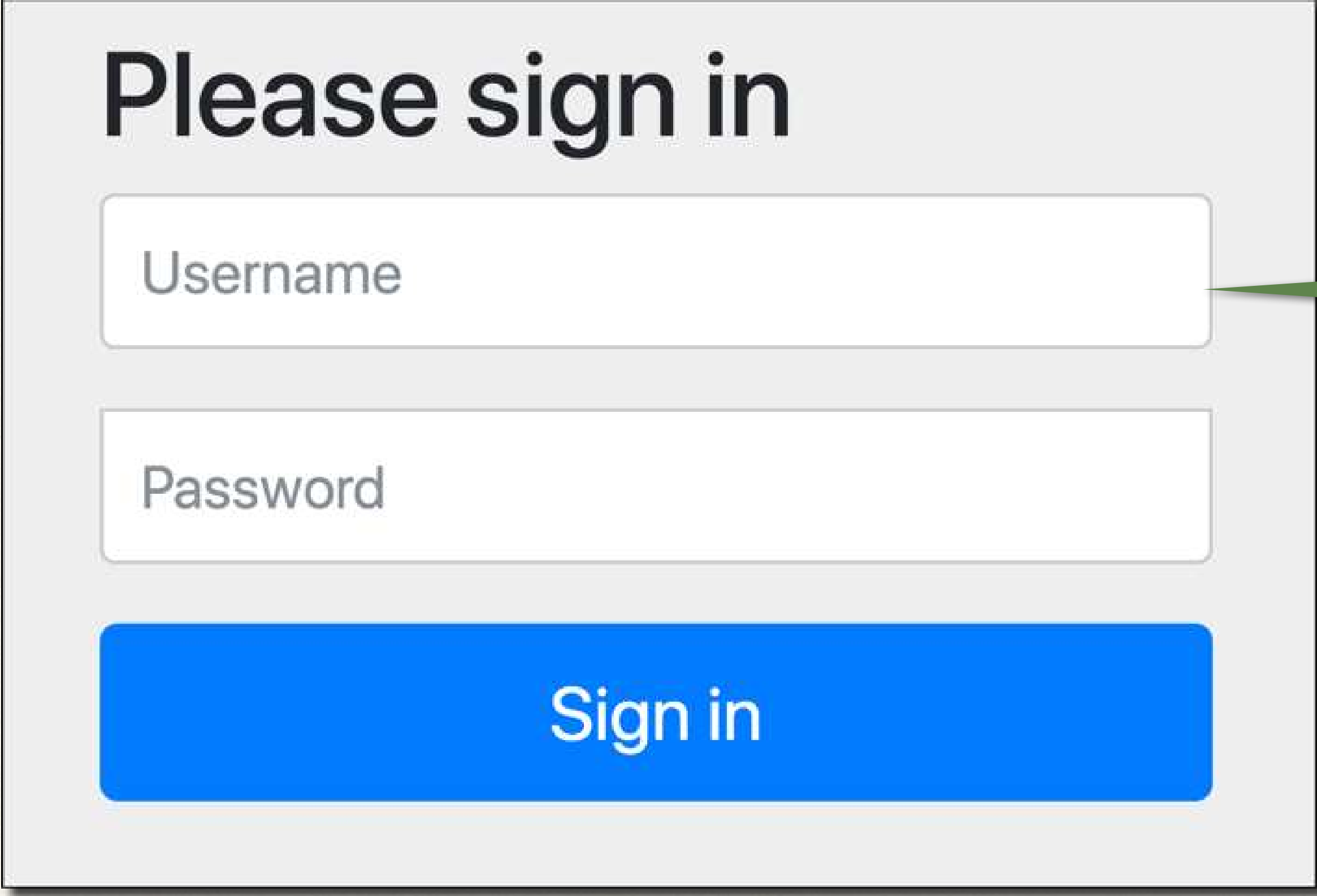
HTTP Basic Authentication



**Built-in Dialog
from browser**

bleh!

Spring Security - Default Login Form



The image shows a default login form for Spring Security. It is a light gray rectangular box with a black border. At the top left, the text "Please sign in" is written in a bold, black, sans-serif font. Below this text are two white input fields with rounded corners and thin gray borders. The first input field is labeled "Username" in a light gray font. The second input field is labeled "Password" in a light gray font. Below the password field is a large, solid blue button with rounded corners, containing the text "Sign in" in white, bold, sans-serif font. To the right of the form, there is a dark green rectangular box with rounded corners. A green line with a triangular arrow points from the right side of the "Username" input field to the left side of the green box. Inside the green box, the text "Good for quick start" is written in white, bold, sans-serif font.

Please sign in

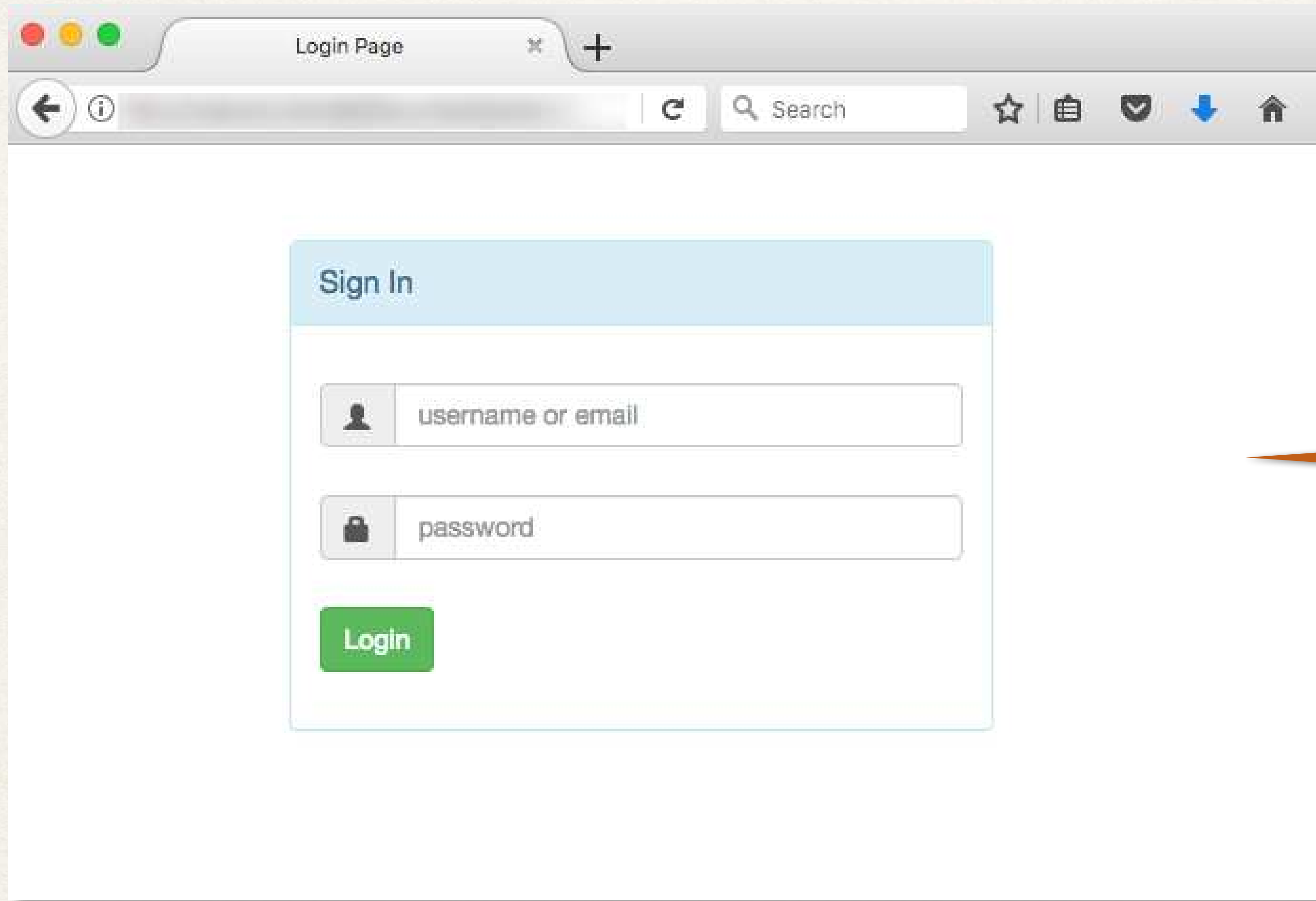
Username

Password

Sign in

Good for quick start

Your Own Custom Login Form



Sign In

username or email

password

Login

**Your own custom
look-and-feel
HTML + CSS**

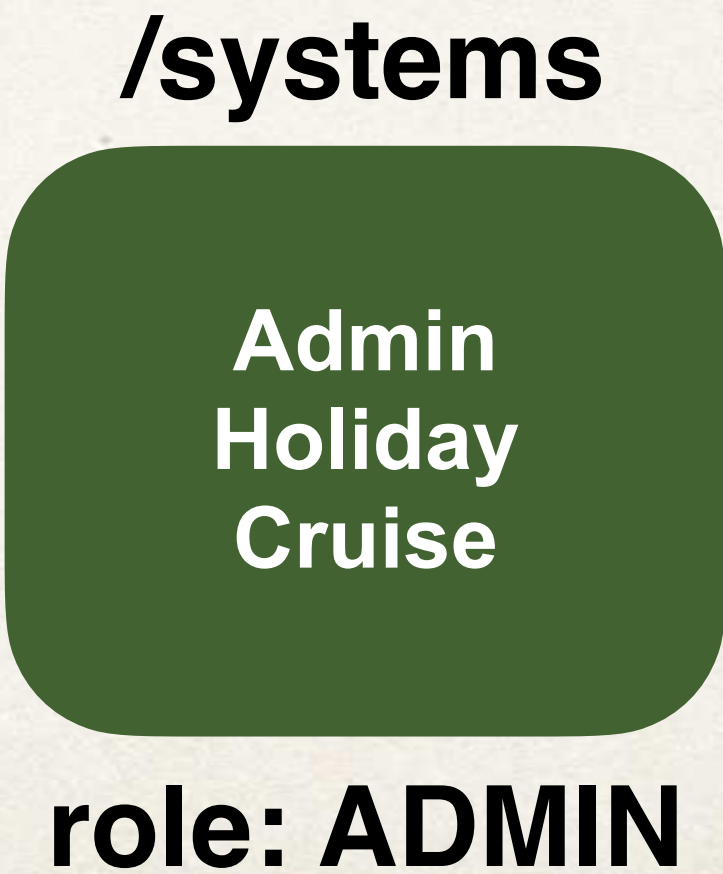
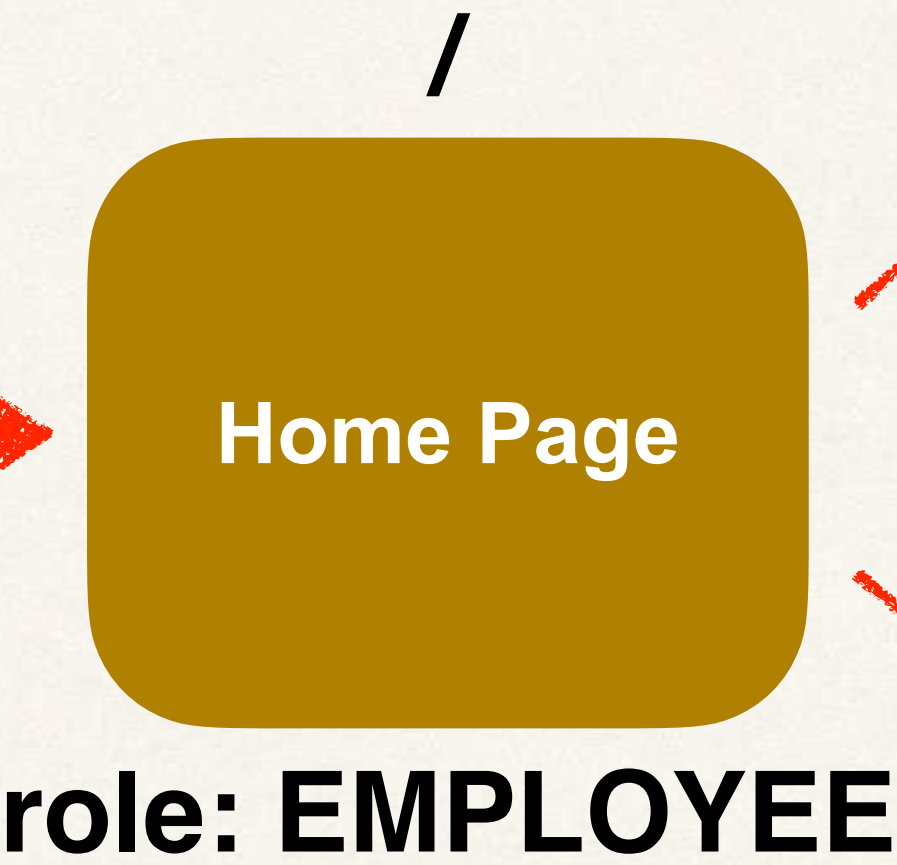
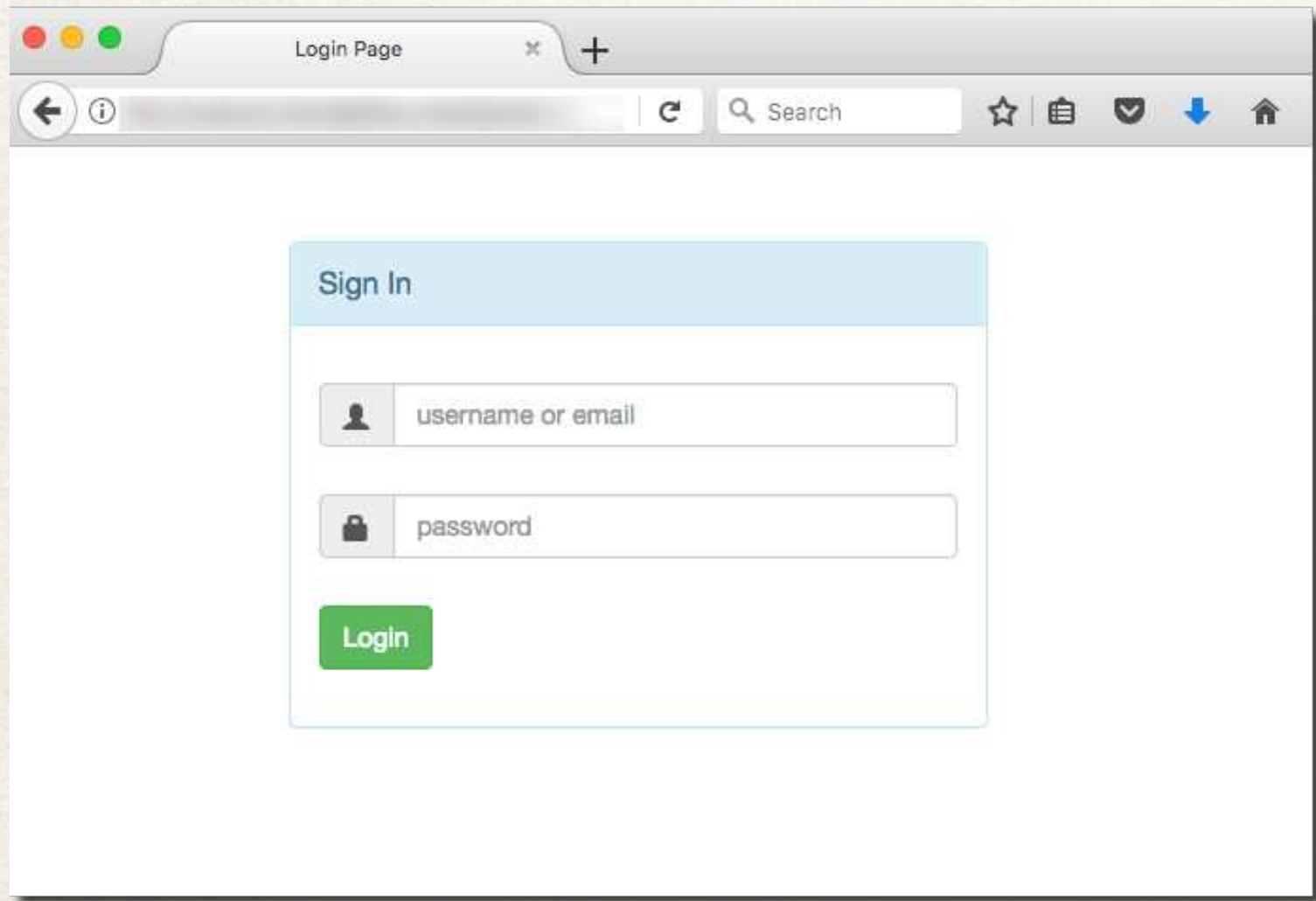
Spring Security Demo



Recap of our Game Plan

- Secure Spring MVC Web Apps
- Develop login pages (default and custom)
- Define users and roles with simple authentication
- Protect URLs based on role
- Hide/show content based on role
- Store users, passwords and roles in DB (plain-text -> encrypted)

Our Example



Spring MVC Security Project Setup



Development Process

Step-By-Step

1. Create project at Spring Initializr website
 1. Add Maven dependencies for Spring MVC Web App, Security, Thymeleaf
2. Develop our Spring controller
3. Develop our Thymeleaf view page

Step 1: Add Maven dependencies for Spring MVC Web App

File: pom.xml

```
...  
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-web</artifactId>  
</dependency>  
  
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-thymeleaf</artifactId>  
</dependency>  
  
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-security</artifactId>  
</dependency>  
  
<dependency>  
  <groupId>org.thymeleaf.extras</groupId>  
  <artifactId>thymeleaf-extras-springsecurity6</artifactId>  
</dependency>  
...
```

Spring MVC web support

Thymeleaf view support

Spring Security support

Thymeleaf Security support

Step 2: Develop our Spring Controller

File: DemoController.java

```
@Controller
public class DemoController {

    @GetMapping("/")
    public String showHome() {

        return "home";
    }

}
```



View name

src/main/resources/templates/home.html

Step 3: Develop our Thymeleaf view page

File: src/main/resources/templates/home.html

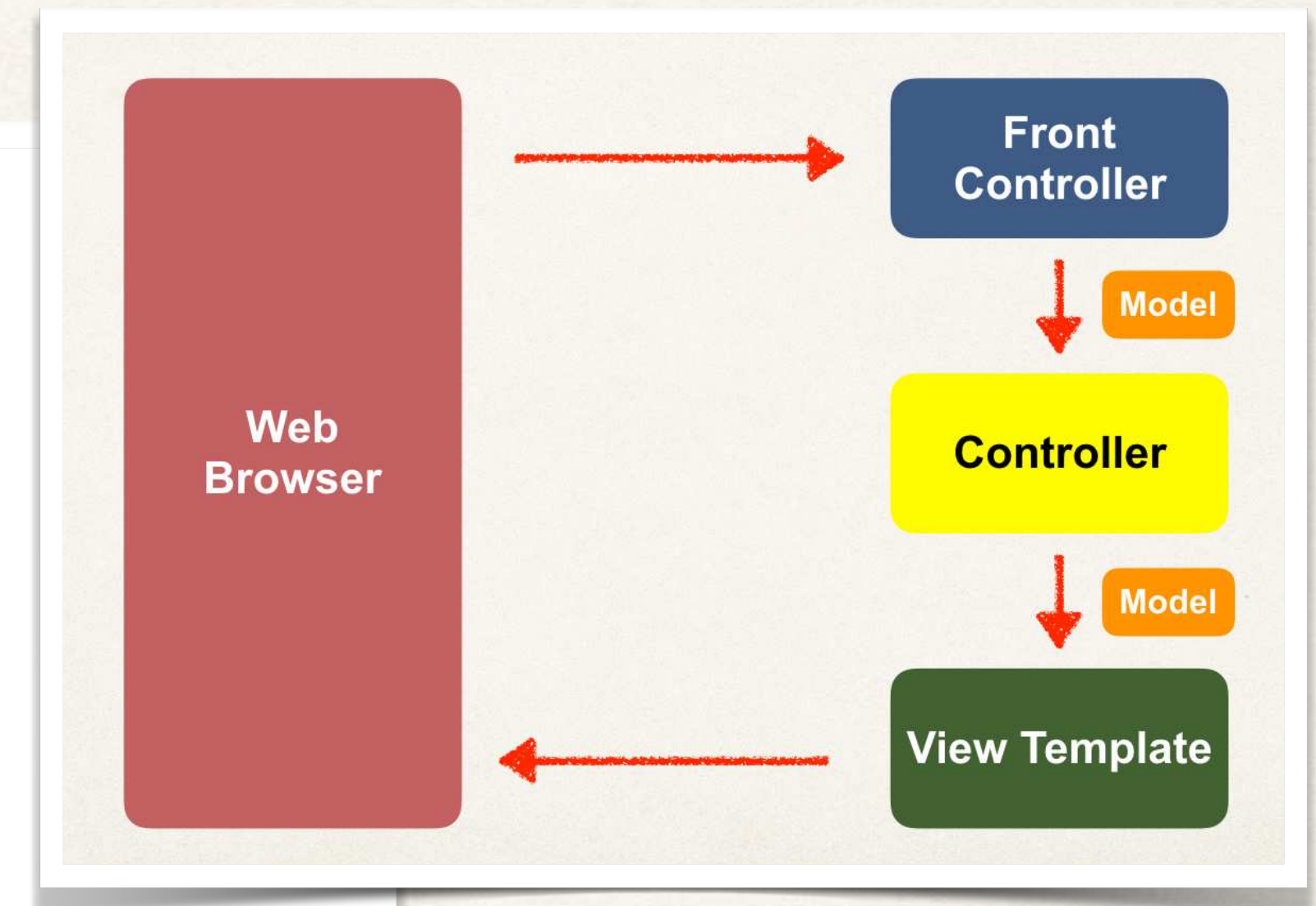
```
<html>
```

```
<body>
```

Welcome to the luv2code company home page!

```
</body>
```

```
</html>
```



Get Fresh Web Session

- Option 1:
 - Start new web browser
- Option 2:
 - In web browser and open new Private / Incognito window

Configuring Basic Security



Our Users

User ID	Password	Roles
john	test123	EMPLOYEE
mary	test123	EMPLOYEE, MANAGER
susan	test123	EMPLOYEE, MANAGER, ADMIN

We can give ANY names
for user roles

Development Process

Step-By-Step

1. Create Spring Security Configuration (@Configuration)
2. Add users, passwords and roles

Step 1: Create Spring Security Configuration

File: DemoSecurityConfig.java

```
import org.springframework.context.annotation.Configuration;

@Configuration
public class DemoSecurityConfig {

    // add our security configurations here ...

}
```


Spring Security Password Storage

- In Spring Security, passwords are stored using a specific format

`{id}encodedPassword`

ID	Description
<code>noop</code>	Plain text passwords
<code>bcrypt</code>	BCrypt password hashing
...	...

Password Example

The encoding
algorithm id

The password

{noop}test123

Let's Spring Security
know the passwords are
stored as plain text (noop)

Step 2: Add users, passwords and roles

File: DemoSecurityConfig.java

```
@Configuration
public class DemoSecurityConfig {

    @Bean
    public InMemoryUserDetailsManager userDetailsManager() {

        UserDetails john = User.builder()
            .username("john")
            .password("{noop}test123")
            .roles("EMPLOYEE")
            .build();

        UserDetails mary = User.builder()
            .username("mary")
            .password("{noop}test123")
            .roles("EMPLOYEE", "MANAGER")
            .build();

        UserDetails susan = User.builder()
            .username("susan")
            .password("{noop}test123")
            .roles("EMPLOYEE", "MANAGER", "ADMIN")
            .build();

        return new InMemoryUserDetailsManager(john, mary, susan);
    }
}
```

User ID	Password	Roles
john	test123	EMPLOYEE
mary	test123	EMPLOYEE, MANAGER
susan	test123	EMPLOYEE, MANAGER, ADMIN

We will add DB support in
later videos
(plaintext and encrypted)

Spring Security - Custom Login Form



Spring Security - Default Login Form

Please sign in

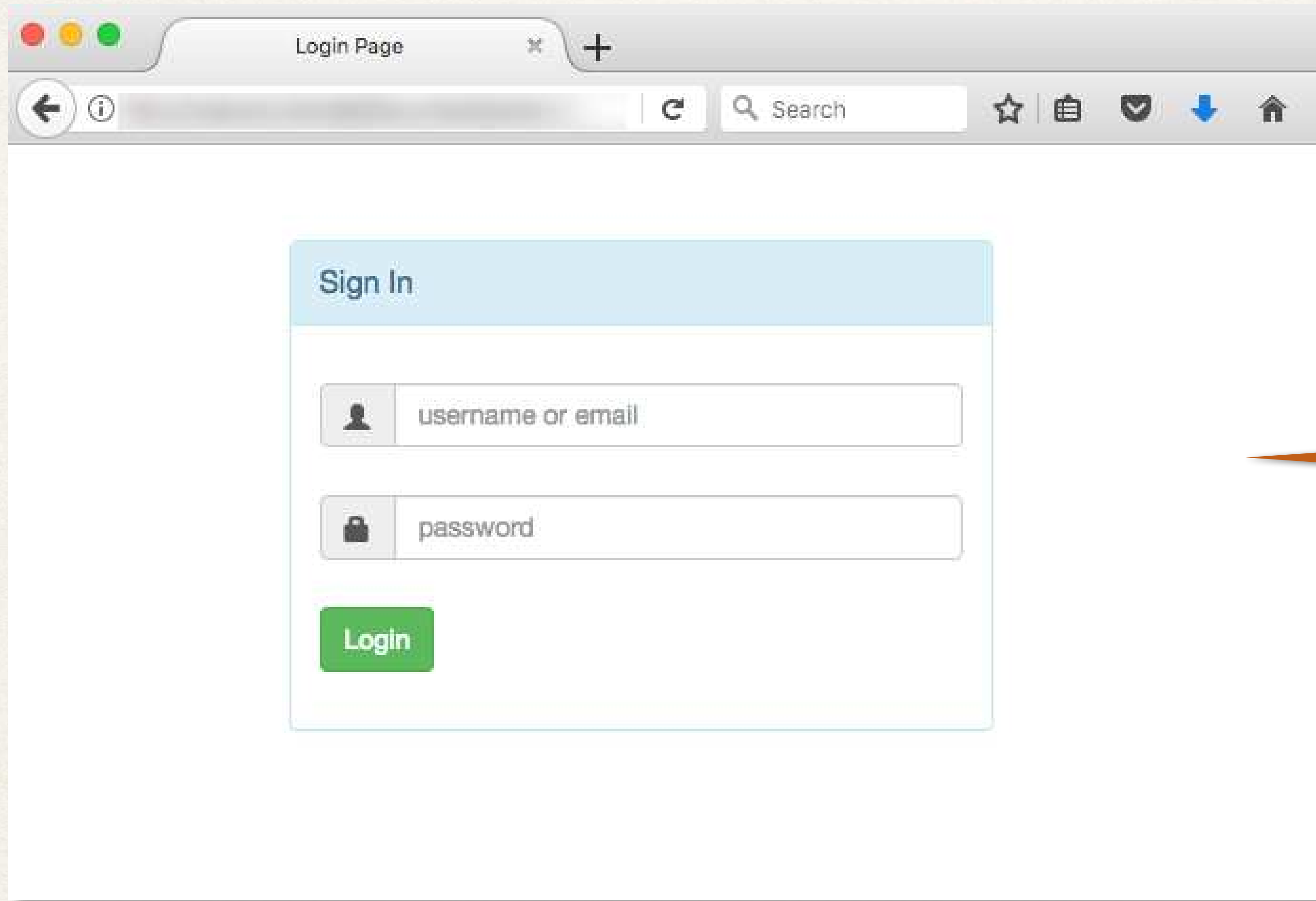
Username

Password

Sign in

Good for quick start

Your Own Custom Login Form



Sign In

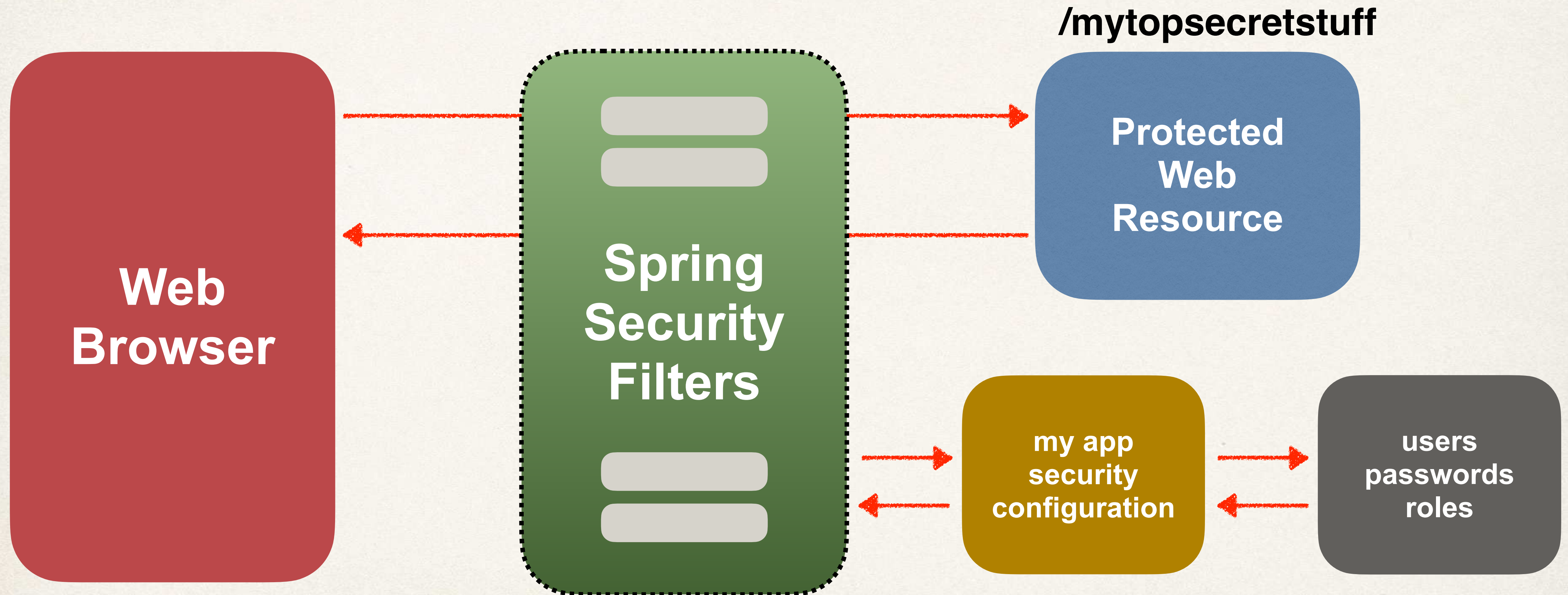
username or email

password

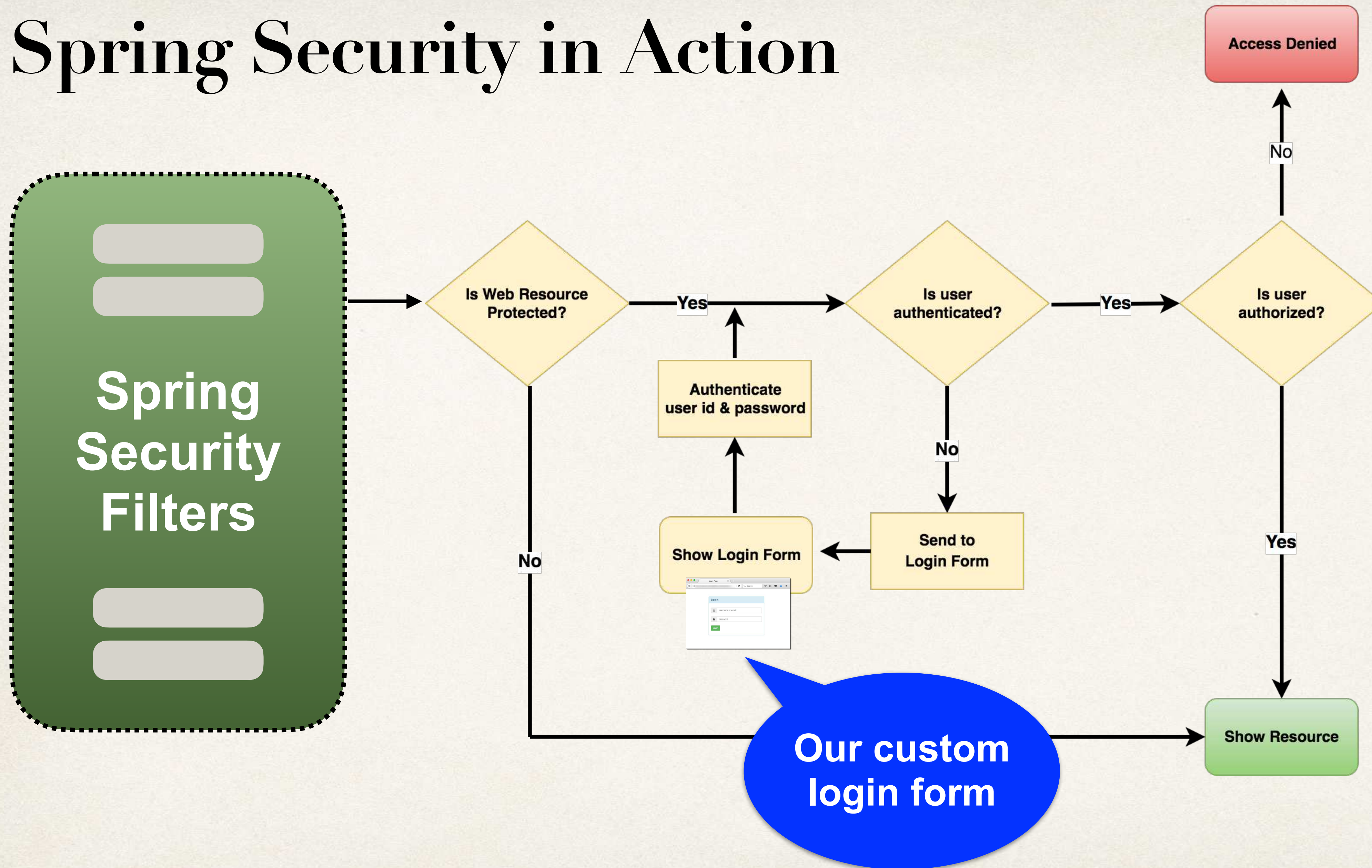
Login

Your own custom
look-and-feel
HTML + CSS

Spring Security Overview



Spring Security in Action



Development Process

Step-By-Step

1. Modify Spring Security Configuration to reference custom login form
2. Develop a Controller to show the custom login form
3. Create custom login form
 - ❖ HTML (CSS optional)

Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(configurer ->
        configurer
            .anyRequest().authenticated()
        )
        .formLogin(form ->
            form
                .loginPage("/showMyLoginPage")
                .loginProcessingUrl("/authenticateTheUser")
                .permitAll()
            );

    return http.build();
}
```

Configure security of web paths
in application, login, logout etc

Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(configurer ->
        configurer
            .anyRequest().authenticated()
        )
        .formLogin(form ->
            form
                .loginPage("/showMyLoginPage")
                .loginProcessingUrl("/authenticateTheUser")
                .permitAll()
            );

    return http.build();
}
```

Any request to the app
must be authenticated
(ie logged in)

Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(configurer ->
        configurer
            .anyRequest().authenticated()
        )
        .formLogin(form ->
            form
                .loginPage("/showMyLoginPage")
                .loginProcessingUrl("/authenticateTheUser")
                .permitAll()
            );

    return http.build();
}
```

We are customizing the form login process

Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(configurer ->
        configurer
            .anyRequest().authenticated()
        )
        .formLogin(form ->
            form
                .loginPage("/showMyLoginPage")
                .loginProcessingUrl("/authenticateTheUser")
                .permitAll()
            );

    return http.build();
}
```

Show our custom form at the
request mapping

“/showMyLoginPage”

Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(configurer ->
        configurer
            .anyRequest().authenticated()
        )
        .formLogin(form ->
            form
                .loginPage("/showMyLoginPage")
                .loginProcessingUrl("/authenticateTheUser")
                .permitAll()
            );

    return http.build();
}
```

Login form should POST data to this URL for processing
(check user id and password)

Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) {

    http.authorizeHttpRequests(configurer ->
        configurer
            .anyRequest().authenticated()
        )
        .formLogin(form ->
            form
                .loginPage("/showMyLoginPage")
                .loginProcessingUrl("/authenticateTheUser")
                .permitAll()
            );

    return http.build();
}
```

You can give ANY values
for this configuration.
Just stay consistent in your app

Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(configurer ->
        configurer
            .anyRequest().authenticated()
        )
        .formLogin(form ->
            form
                .loginPage("/showMyLoginPage")
                .loginProcessingUrl("/authenticateTheUser")
                .permitAll()
            );

    return http.build();
}
```

Allow everyone to see login page.
No need to be logged in.

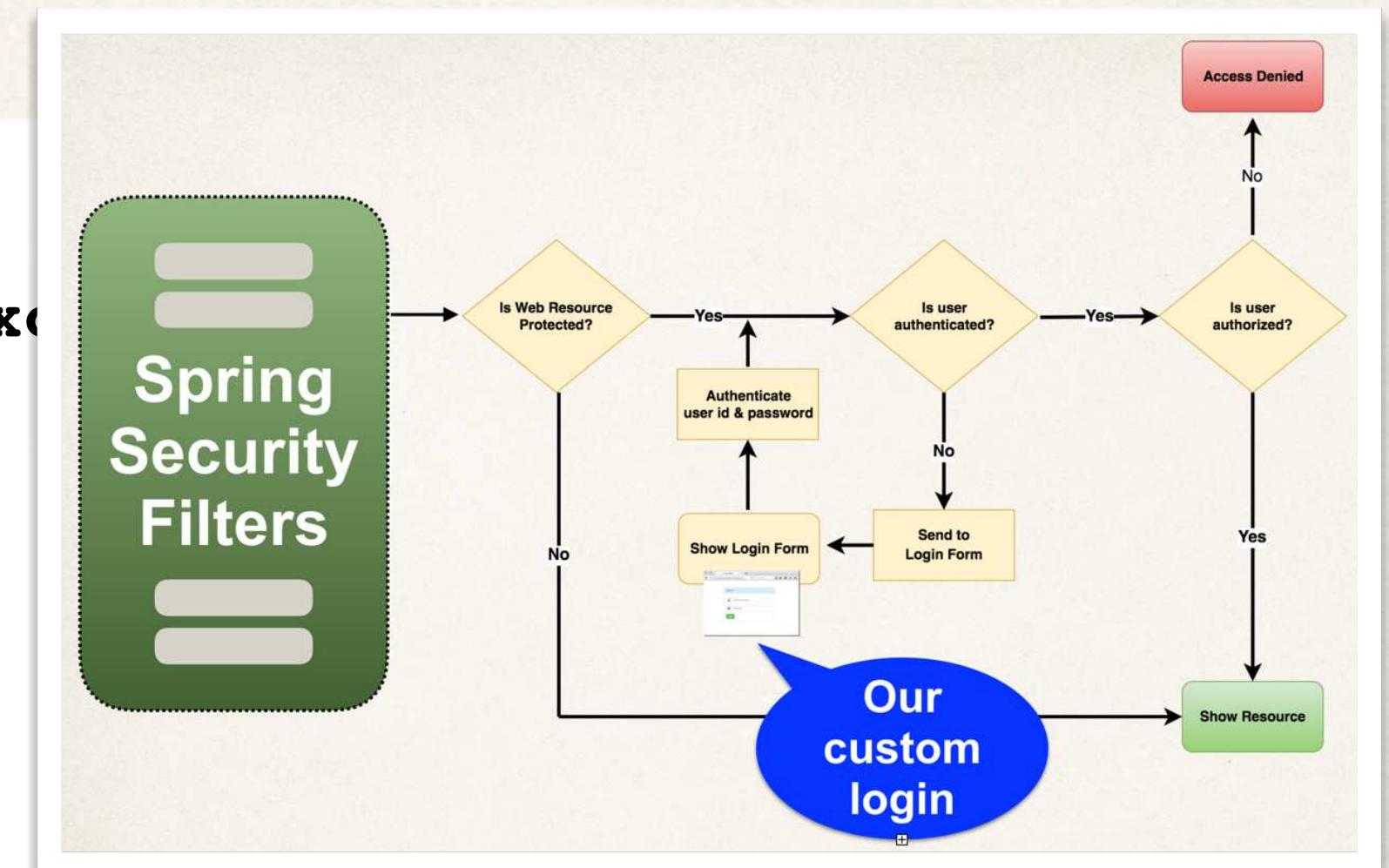
Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(configurer ->
        configurer
            .anyRequest().authenticated()
    )
    .formLogin(form ->
        form
            .loginPage("/showMyLoginPage")
            .loginProcessingUrl("/authenticateTheUser")
            .permitAll()
    );

    return http.build();
}
```



Allow everyone to see login page.
No need to be logged in.

Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws
    http.authorizeHttpRequests(configurer ->
        configurer
            .anyRequest().authenticated()
        )
        .formLogin(form ->
            form
                .loginPage("/showMyLoginPage")
                .loginProcessingUrl("/authenticateTheUser")
                .permitAll()
            );

    return http.build();
}
```

TO DO

*We need to create a controller
for this request mapping*

"/showMyLoginPage"

Step 1: Modify Spring Security Configuration

File: DemoSecurityConfig.java

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(configurer ->
        configurer
            .anyRequest().authenticated()
        )
        .formLogin(form ->
            form
                .loginPage("/showMyLoginPage")
                .loginProcessingUrl("/authenticateTheUser")
                .permitAll()
            );

    return http.build();
}
```

No Controller Request Mapping
required for this.
We get this for free :-)

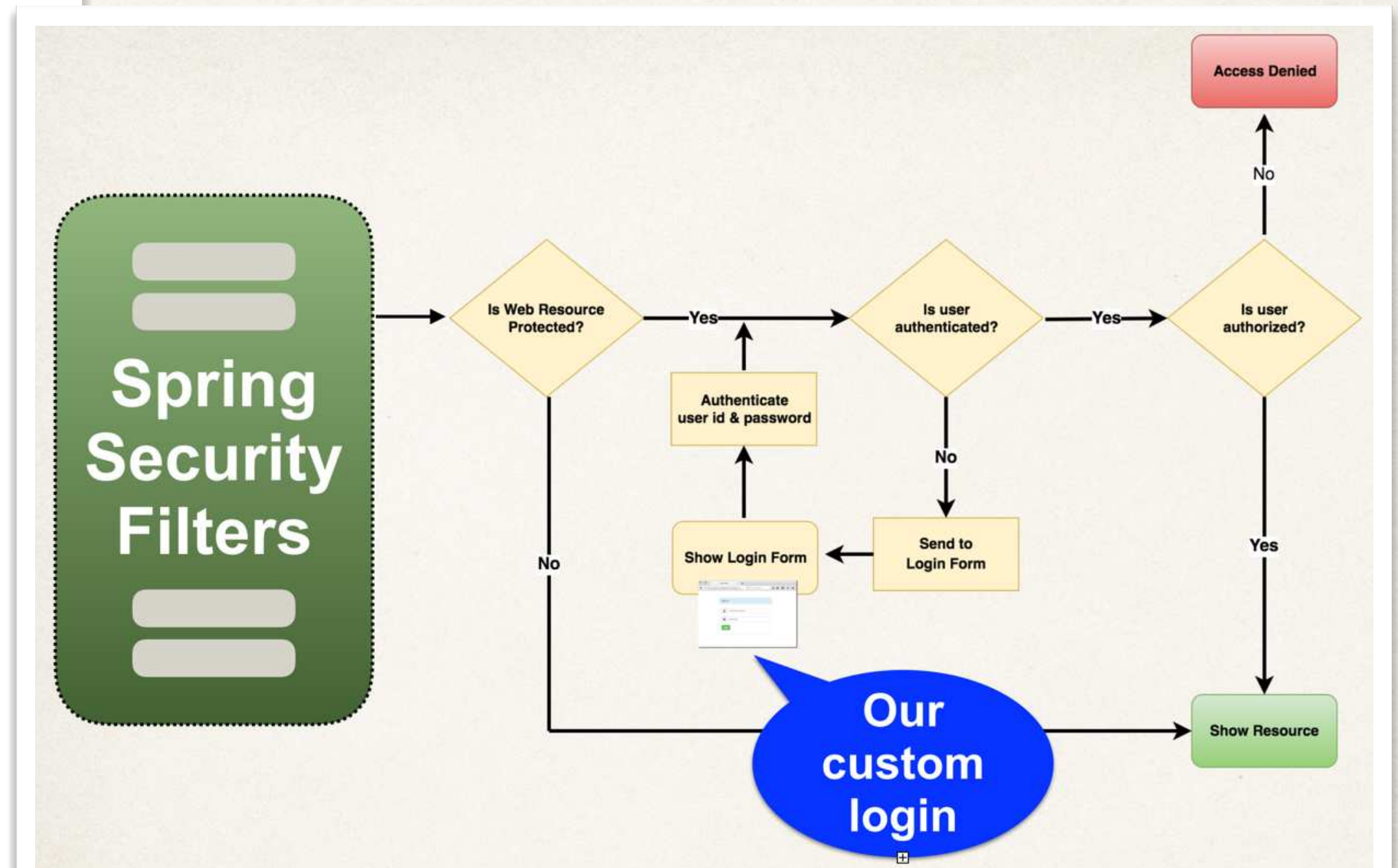
Step 2: Develop a Controller to show the custom login form

File: LoginController.java

```
@Controller
public class LoginController {

    @GetMapping("/showMyLoginPage")
    public String showMyLoginPage() {

        return "plain-login";
    }
}
```



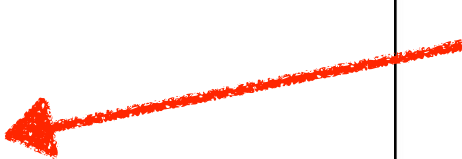
Step 2: Develop a Controller to show the custom login form

File: LoginController.java

```
@Controller
public class LoginController {

    @GetMapping("/showMyLoginPage")
    public String showMyLoginPage() {

        return "plain-login";
    }
}
```



```
.loginPage("/showMyLoginPage")
.loginProcessingUrl("/authenticateTheUser")
.permitAll()
```

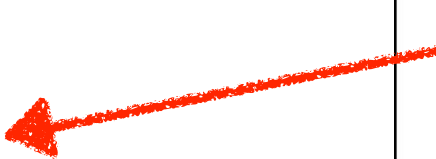

Step 2: Develop a Controller to show the custom login form

File: LoginController.java

```
@Controller
public class LoginController {

    @GetMapping("/showMyLoginPage")
    public String showMyLoginPage() {

        return "plain-login";
    }
}
```



```
.loginPage("/showMyLoginPage")
.loginProcessingUrl("/authenticateTheUser")
.permitAll()
```

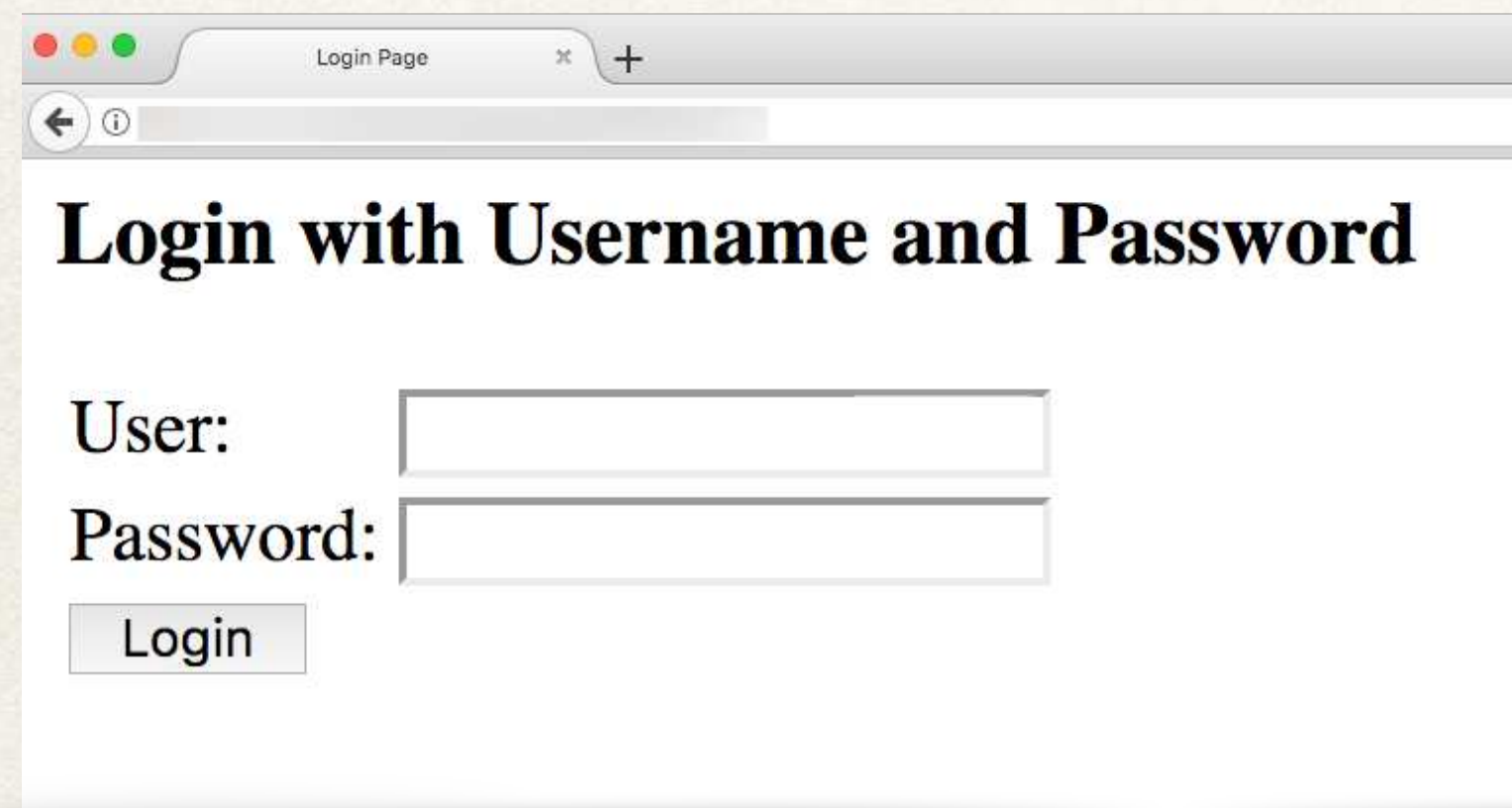
View name

TO DO
We need to create this file

src/main/resources/templates/plain-login.html

Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**



Login Page

Login with Username and Password

User:

Password:

Login

You can give ANY values
for this configuration.

Just stay consistent in your app

/authenticateTheUser

Spring
Security
Filters

```
.loginPage("/showMyLoginPage")  
.loginProcessingUrl("/authenticateTheUser")  
.permitAll()
```


Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**
- Login processing URL will be handled by Spring Security Filters
- You get it for free ... no coding required

This is
Spring Security magic ...
LOL



Step 3: Create custom login form

- Send data to login processing URL: **/authenticateTheUser**
- Must **POST** the data

```
<form action="#" th:action="@{/authenticateTheUser}"  
      method="POST">  
    ...  
</form>
```

```
.loginPage("/showMyLoginPage")  
.loginProcessingUrl("/authenticateTheUser")  
.permitAll()
```



Step 3: Create custom login form

- Spring Security defines default names for login form fields
 - User name field: **username**
 - Password field: **password**

Spring Security Filters
will read the form data and
authenticate the user

User name: `<input type="text" name="username" />`

Password: `<input type="password" name="password" />`

Step 3: Create custom login form

Pull It All
Together

File: src/main/resources/templates/plain-login.html

```
...  
<form action="#" th:action="@{/authenticateTheUser}"  
      method="POST">  
  
  <p>  
    User name: <input type="text" name="username" />  
  </p>  
  
  <p>  
    Password: <input type="password" name="password" />  
  </p>  
  
  <input type="submit" value="Login" />  
  
</form>  
...
```


Step 3: Create custom login form

Pull It All Together

File: src/main/resources/templates/plain-login.html

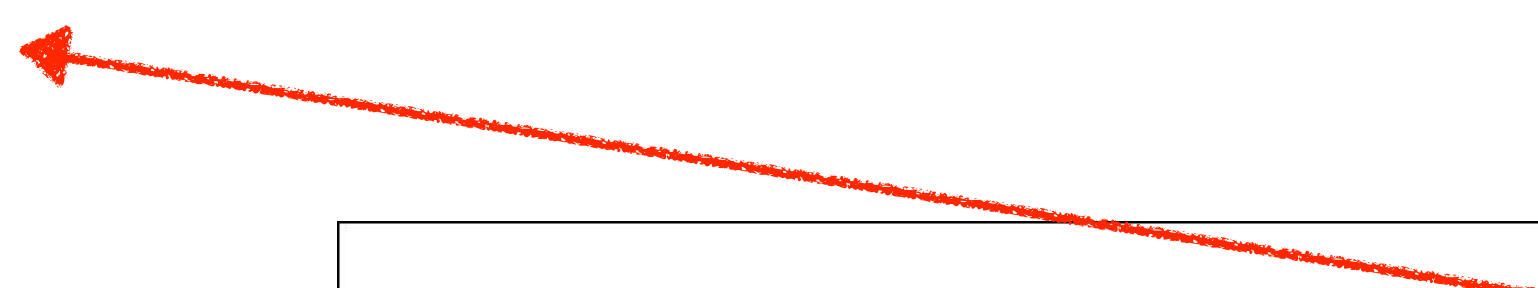
```
...
<form action="#" th:action="@{/authenticateTheUser}"
      method="POST">

  <p>
    User name: <input type="text" name="username" />
  </p>

  <p>
    Password: <input type="password" name="password" />
  </p>

  <input type="submit" value="Login" />

</form>
...
```



```
.loginPage("/showMyLoginPage")
.loginProcessingUrl("/authenticateTheUser")
.permitAll()
```


Step 3: Create custom login form

Pull It All Together

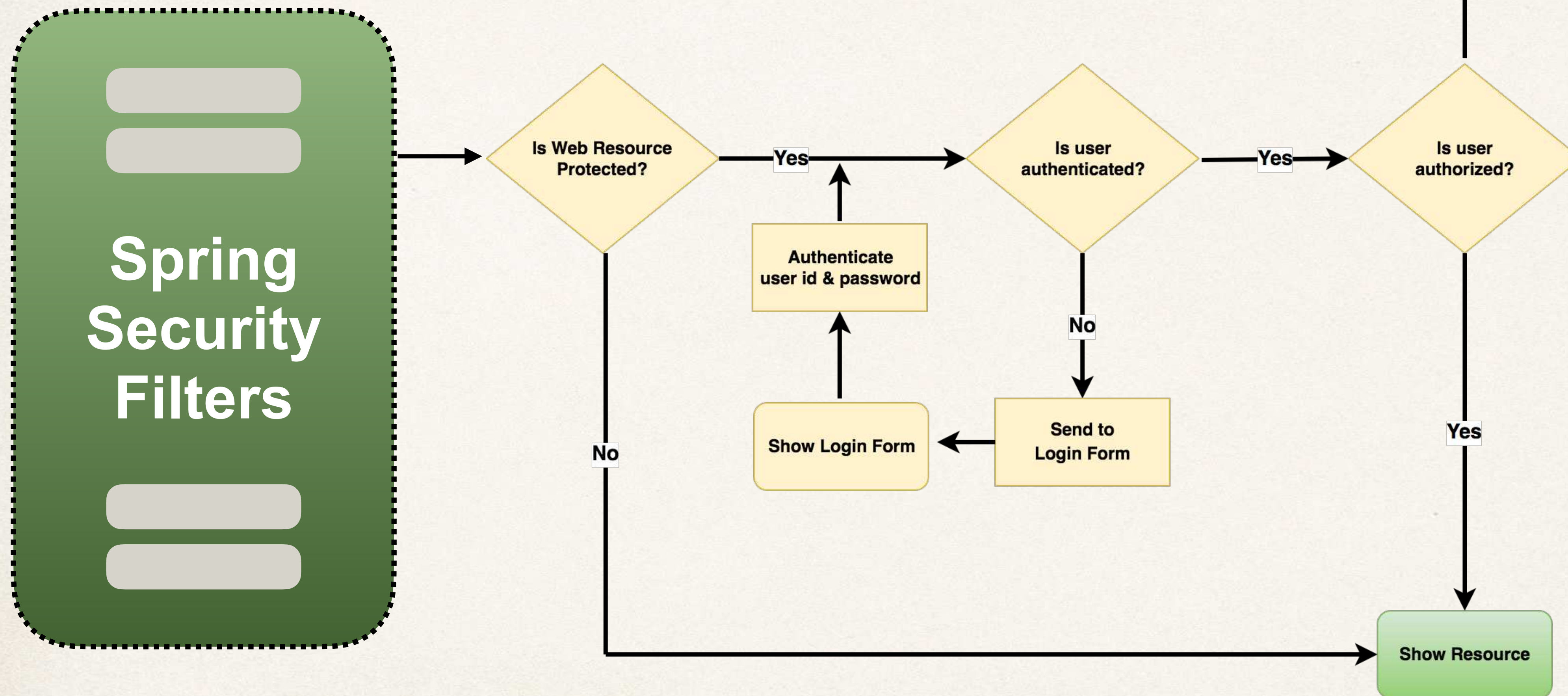
File: src/main/resources/templates/plain-login.html

```
...  
<form action="#" th:action="@{/authenticateTheUser}"  
      method="POST">  
  
  <p>  
    User name: <input type="text" name="username" />  
  </p>  
  
  <p>  
    Password: <input type="password" name="password" />  
  </p>  
  
  <input type="submit" value="Login" />  
  
</form>  
...
```

`.loginPage("/showMyLoginPage")`
`.loginProcessingUrl("/authenticateTheUser")`
`.permitAll()`



Spring Security in Action



More Info on Context Path

What is this???

The @ symbol

```
<form action="#" th:action="@{/authenticateTheUser}"  
      method="POST">
```

...

More Info on Context Path

What is "Context Root"

Context Path
is same thing as
Context Root

The root path for your
web application

Context Root: my-ecommerce-app

<http://localhost:8080/my-ecommerce-app>

More Info on Context Path

Gives us access to
context path dynamically

```
<form action="#" th:action="@{/authenticateTheUser}"  
      method="POST">
```

...

Why use Context Path?

Best Practice

- Allows us to dynamically reference context path of application
- Helps to keep links relative to application context path
- If you change context path of app, then links will still work
- Much better than hard-coding context path ...

Spring Security - Show Login Error



No error message for failed login - What???

My Custom Login Form

User name:

Password:

Login

No error message???

We need an error message ... ASAP!!!

My Custom Login Form

Sorry! You entered invalid username/password.

User name:

Password:

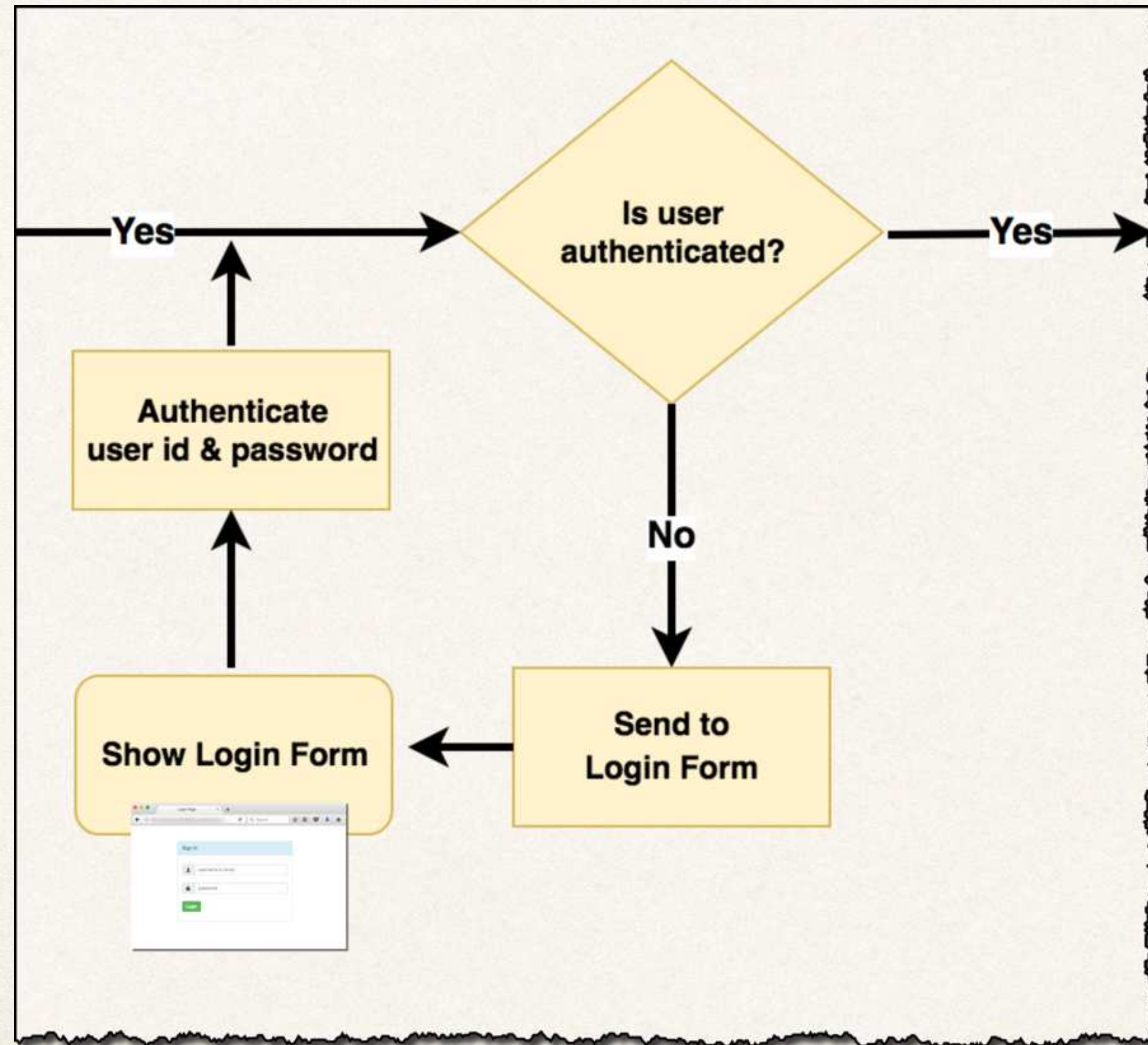
Login

**We'll add this
support**

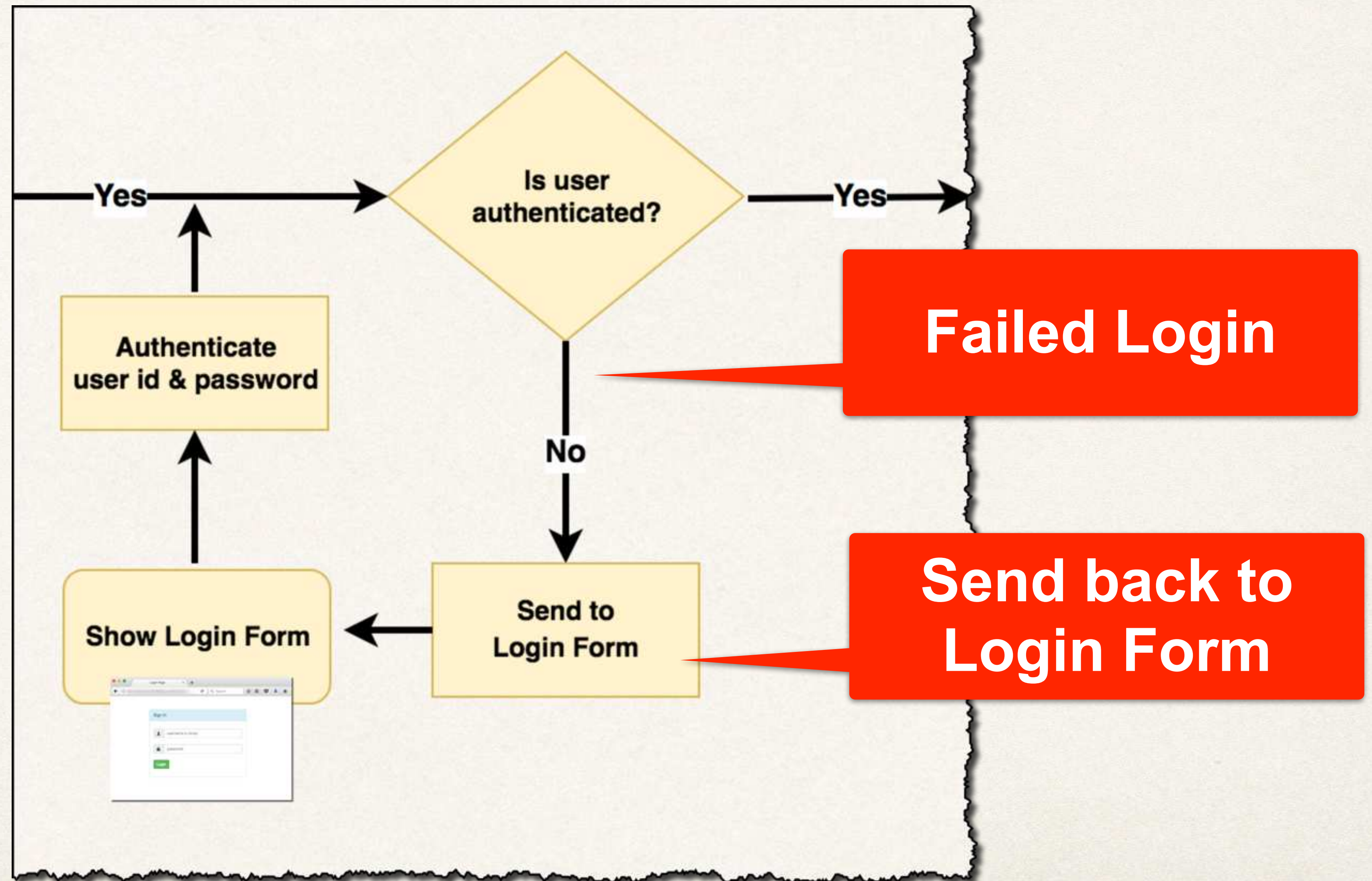
Failed Login

- When login fails, by default Spring Security will ...
- Send user back to your login page
- Append an error parameter: **?error**

Failed Login

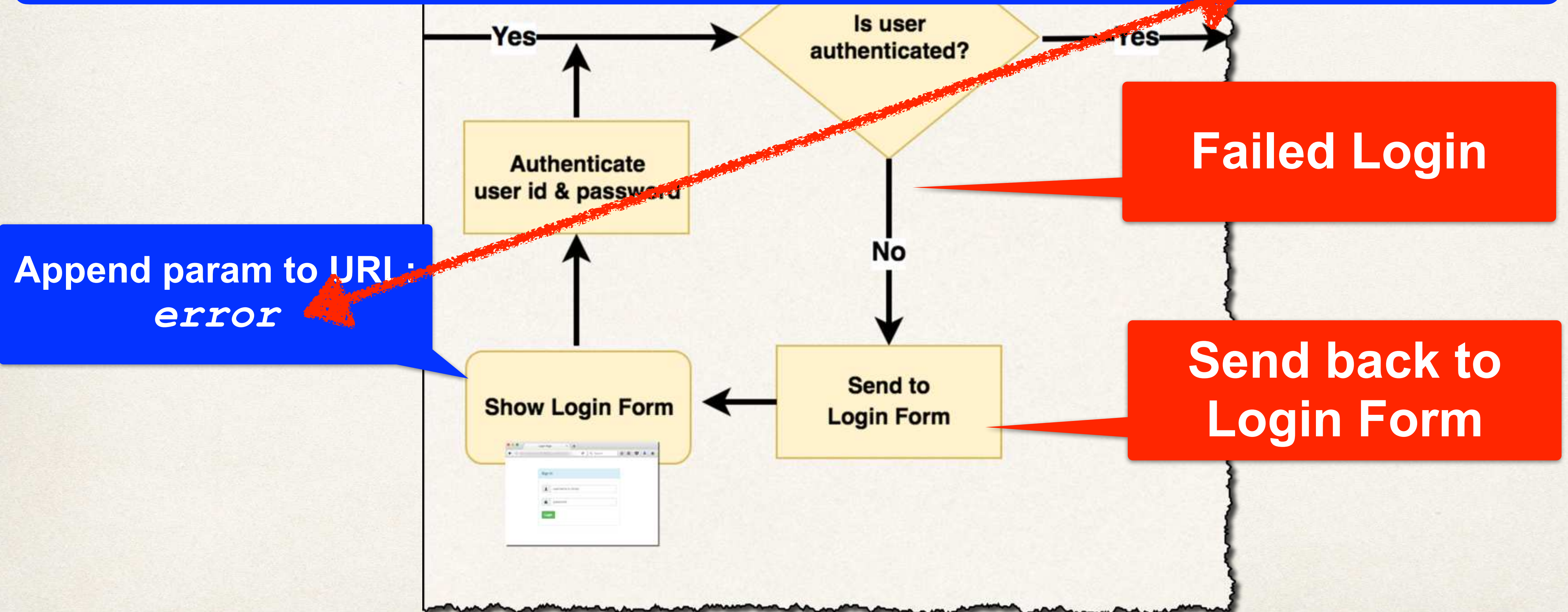


Failed Login

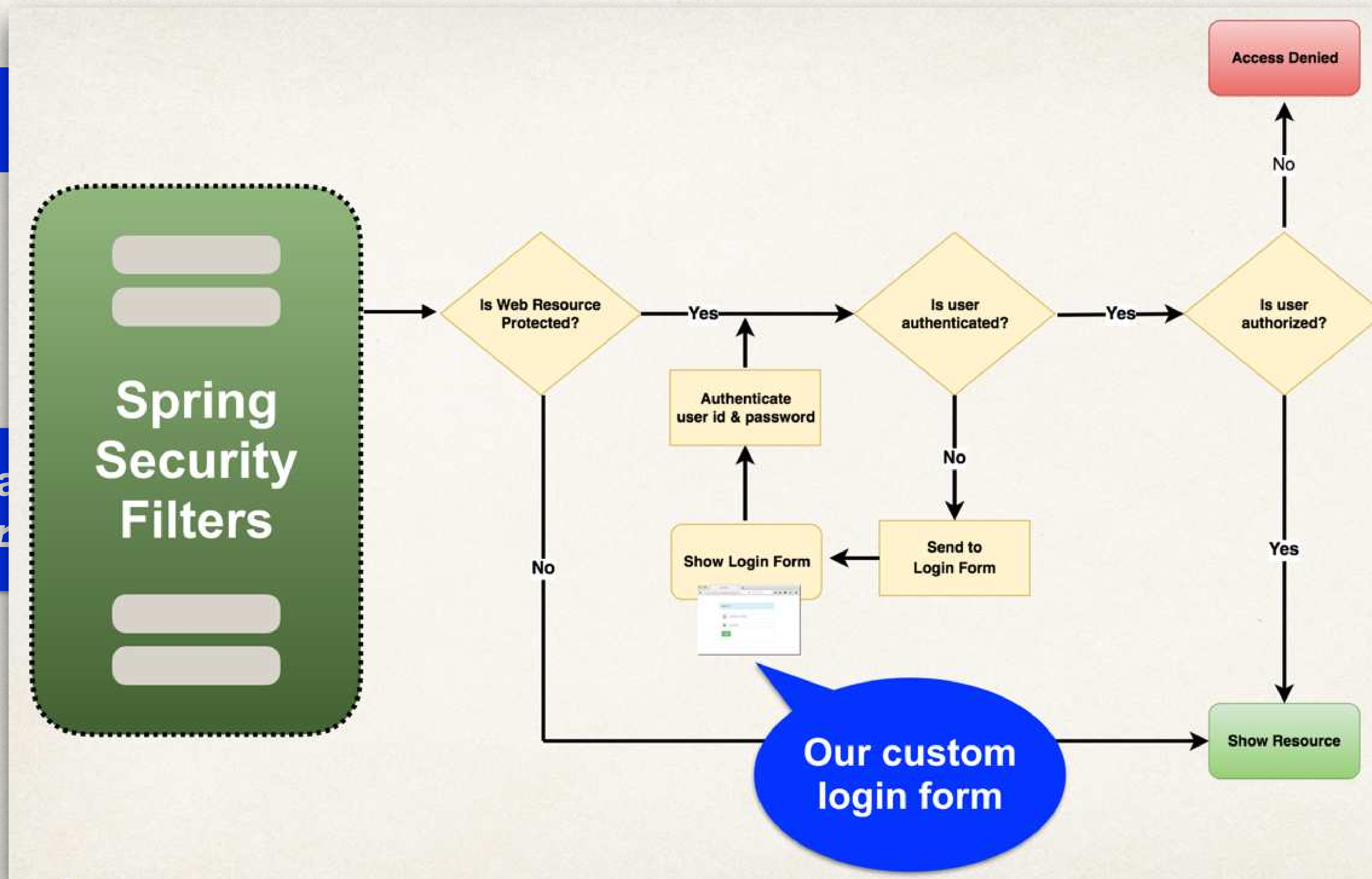


Failed Login

`http://localhost:8080/myapp/showMyLoginPage?error`



Failed Login



Development Process

Step-By-Step

1. Modify custom login form

1. Check the **error** parameter

2. If **error** parameter exists, show an error message

Step 1: Modify form - check for error

File: src/main/resources/templates/plain-login.html

If error param
then show message

```
...  
<form ...>  
  
  <div th:if="{param.error}">  
  
    <i>Sorry! You entered invalid username/password.</i>  
  
  </div>  
  
  User name: <input type="text" name="username" />  
  Password: <input type="password" name="password" />  
  
</form>  
...
```


Step 1: Modify form - check for error

File: src/main/resources/templates/plain-login.html

```
...  
<form ...>
```

```
<div th:if="{param.error}">
```

```
<i>Sorry! You entered invalid username/password.</i>
```

```
</div>
```

```
User name: <input type="text" name="username" />
```

```
Password: <input type="password" name="password" />
```

If error param
then show message

<http://localhost:8080/myapp/showMyLoginPage?error>

Spring Security - Bootstrap Login Page



Let's Make our Login Form Beautiful

Before

My Custom Login Form

User name:

Password:

After

Sign In

username

password

Bootstrap

Bootstrap

- Bootstrap is a web framework that includes CSS styles and JavaScript
- Focused on front-end UI
- For our project, you do not need any experience with Bootstrap :-)

Our Game Plan

- My friend, Alan, has helped us out
 - He created an HTML login form using Bootstrap
- We'll modify the form to use Spring Security for our project

Fancy Login Form

Sign In

username

...

password

...

Login

Development Process

Step-By-Step

1. Modify form to point to our login processing URL
2. Verify form fields for username and password
3. Change our controller to use our new Bootstrap login form

Bootstrap Tutorials and Docs

- There are tons of free tutorials online: “bootstrap tutorial”

www.w3schools.com/bootstrap

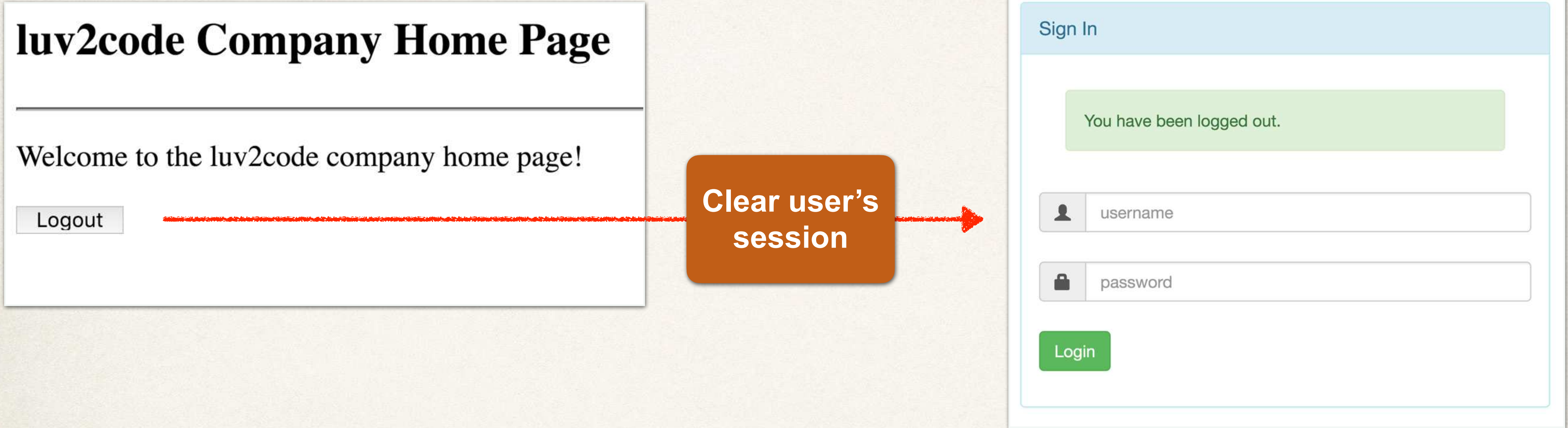
- Bootstrap Official Documentation

www.getbootstrap.com/docs

Spring Security - Logout



Logging Out



Development Process

Step-By-Step

1. Add logout support to Spring Security Configuration
2. Add logout button to home page
3. Update login form to display “logged out” message

Step 1: Add Logout support to Spring Security Configuration

File: DemoSecurityConfig.java

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(configurer ->
        configurer
            .anyRequest().authenticated()
    )
    .formLogin(form ->
        form
            .loginPage("/showMyLoginPage")
            .loginProcessingUrl("/authenticateTheUser")
            .permitAll()
    )
    .logout(logout -> logout.permitAll()
    );

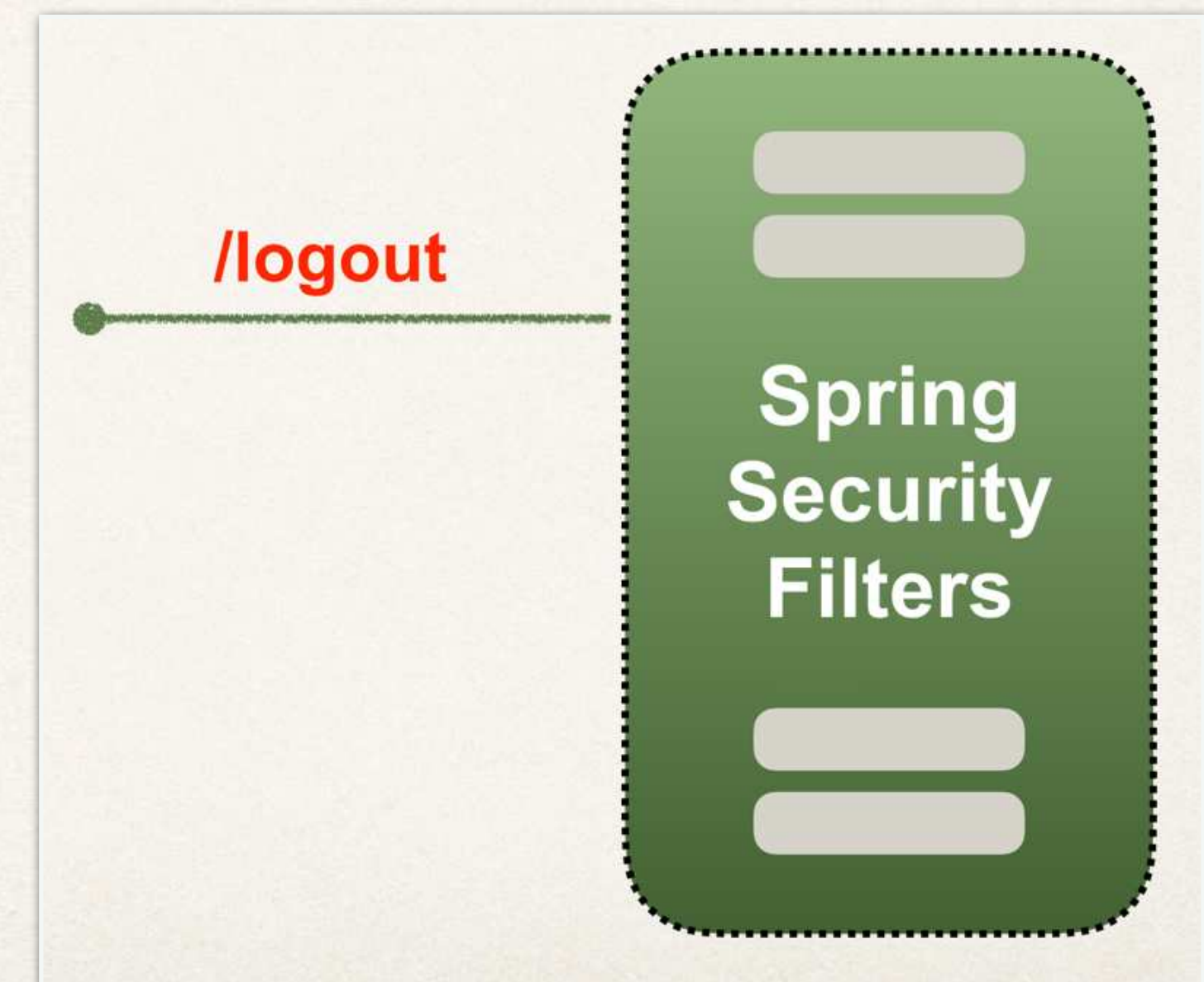
    return http.build();
}
```

Add logout support
for default URL
/logout

Step 2: Add logout button

- Send data to default logout URL: **/logout**
- Logout URL will be handled by Spring Security Filters
- You get it for free ... no coding required

This is
Spring Security magic ...
LOL



Step 2: Add logout button

- Send data to default logout URL: **/logout**
- By default, must use **POST** method

```
<form action="#" th:action="@{/logout}" method="POST">  
    <input type="submit" value="Logout" />  
</form>
```

GET method is disabled
by default

Logout process

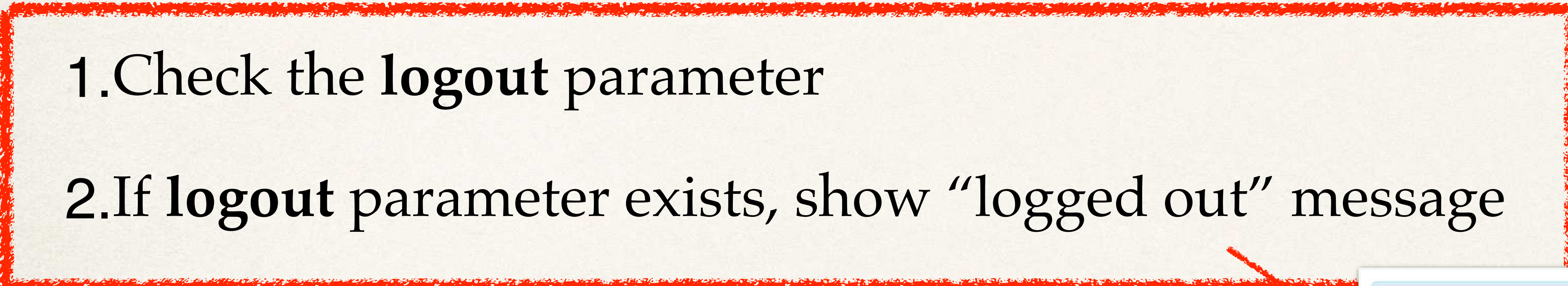
- When a logout is processed, by default Spring Security will ...
- Invalidate user's HTTP session and remove session cookies, etc
- Send user back to your login page
- Append a logout parameter: **?logout**

Step 3: Update login form to display “logged out” message

1. Update login form

1. Check the **logout** parameter

2. If **logout** parameter exists, show “logged out” message



The diagram illustrates the implementation of the 'logged out' message. A red hand-drawn box encloses the two instructions. A red arrow points from the second instruction, 'If logout parameter exists, show “logged out” message', to a 'Sign In' form. The form has a light blue header with the text 'Sign In'. Below the header is a green message box containing the text 'You have been logged out.'. Underneath the message box are two input fields: one for 'username' with a user icon and one for 'password' with a lock icon. At the bottom of the form is a green 'Login' button.

Modify Login form - check for "logout"

File: src/main/resources/templates/plain-login.html

```
...
<form ... th:action="..." method="...">

  <div th:if="{param.logout}">

    <i>You have been logged out.</i>

  </div>

  User name: <input type="text" name="username" />
  Password: <input type="password" name="password" />

</form>
...
```

If logout param then
show message

Modify Login form - check for "logout"

File: src/main/resources/templates/plain-login.html

```
...  
<form ... th:action="..." method="...">
```

```
<div th:if="{param.logout}">
```

```
<i>You have been logged out.</i>
```

```
</div>
```

```
User name: <input type="text" name="username" />
```

```
Password: <input type="password" name="password" />
```

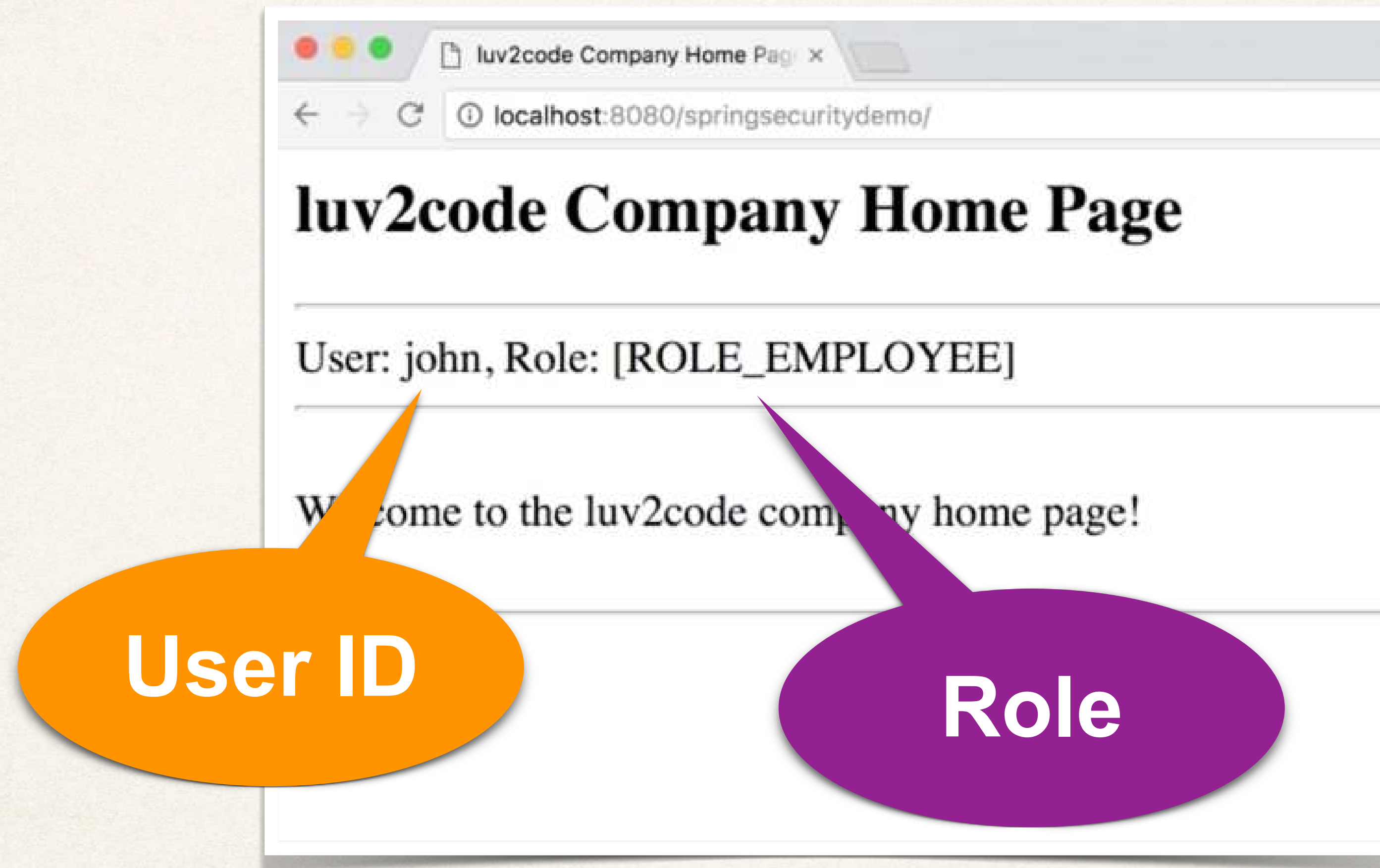
If logout param then
show message

<http://localhost:8080/showMyLoginPage?logout>

Display User ID and Roles



Display User ID and Roles



Spring Security

- Spring Security provides support for accessing user id and roles

Development Process

Step-By-Step

1. Display User ID

2. Display User Roles

Step 1: Display User ID

File: home.html

...

User:



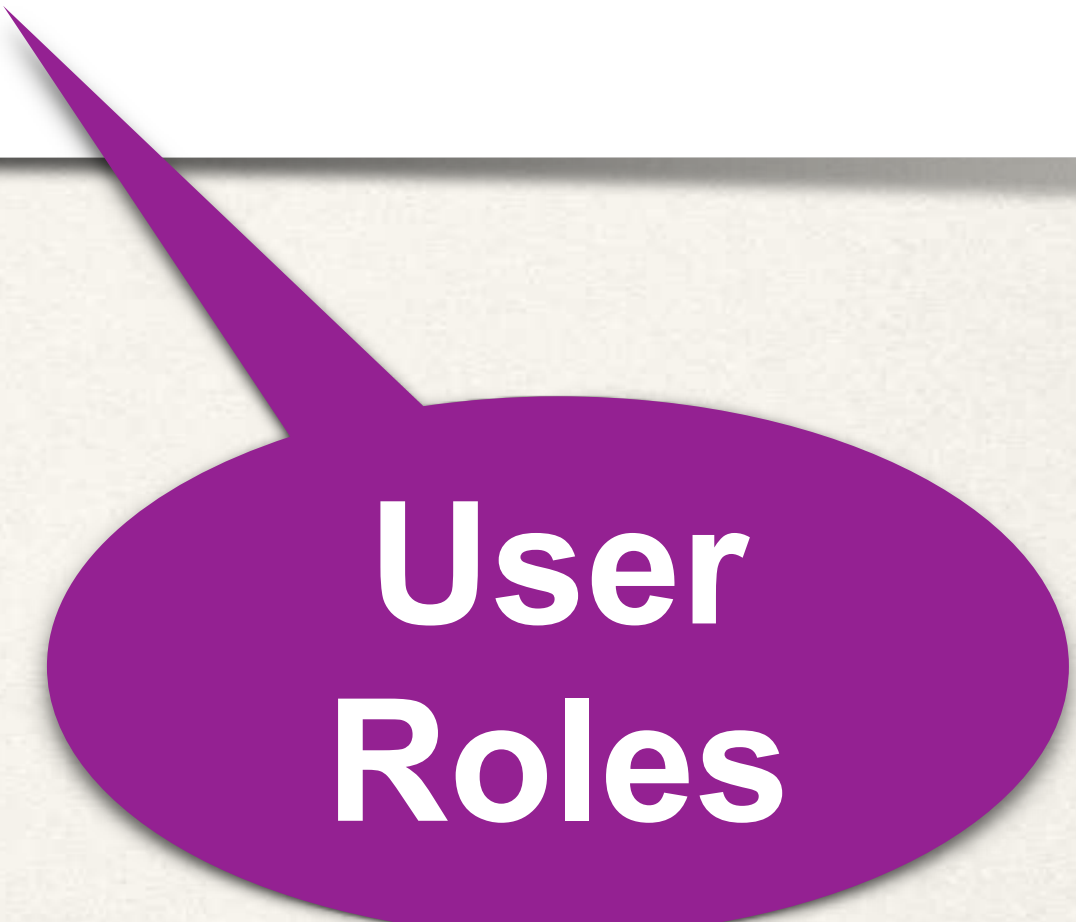
User ID

Step 2: Display User Roles

File: home.html

...

Role(s) :

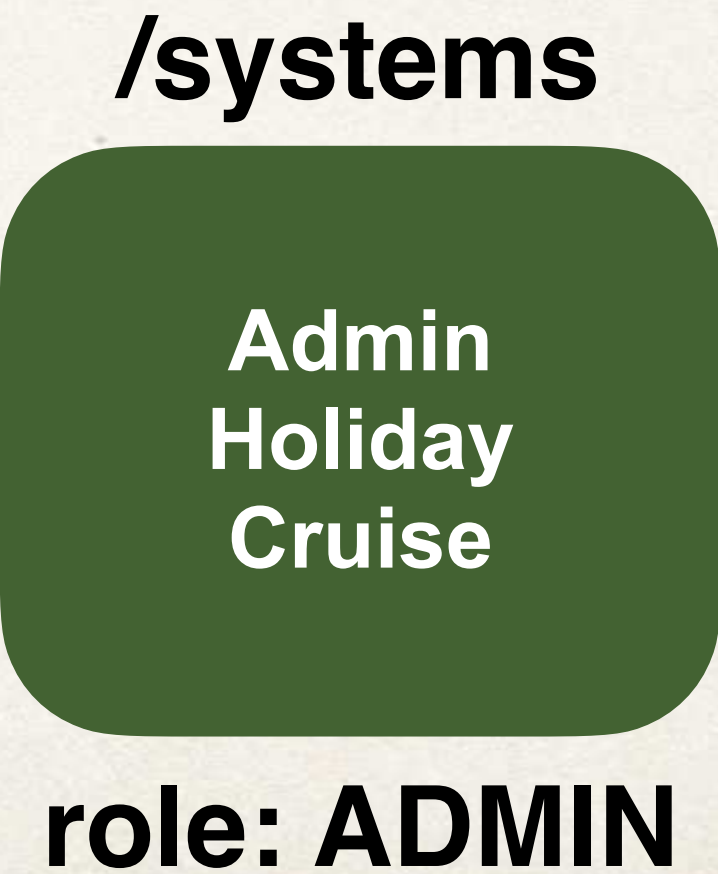
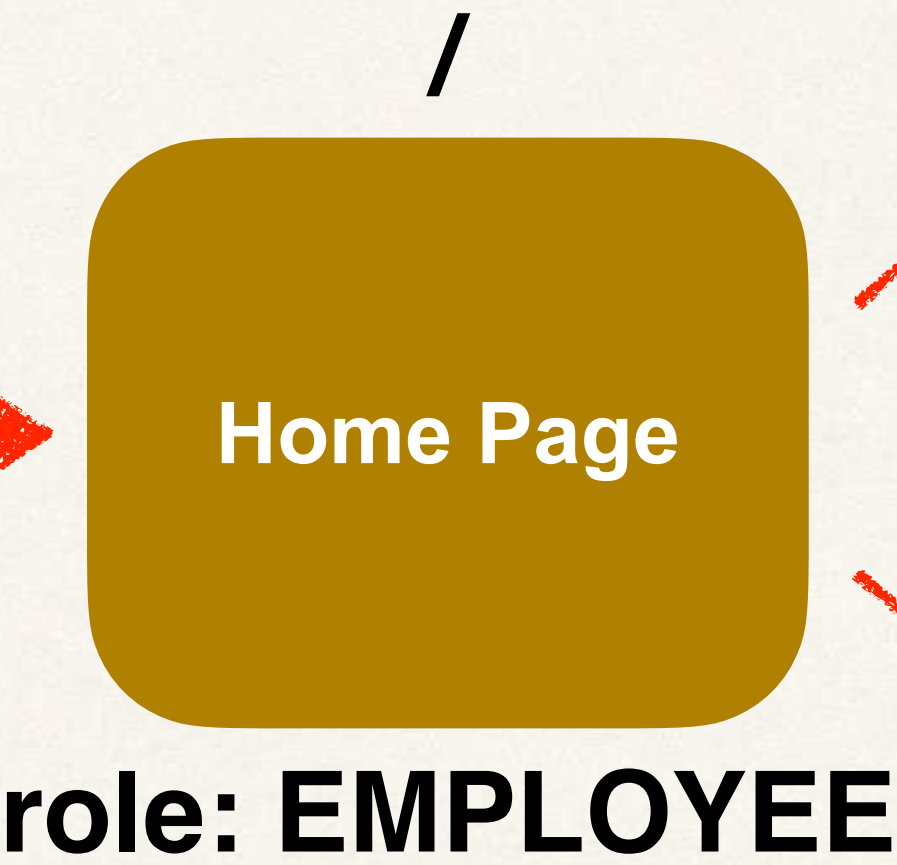
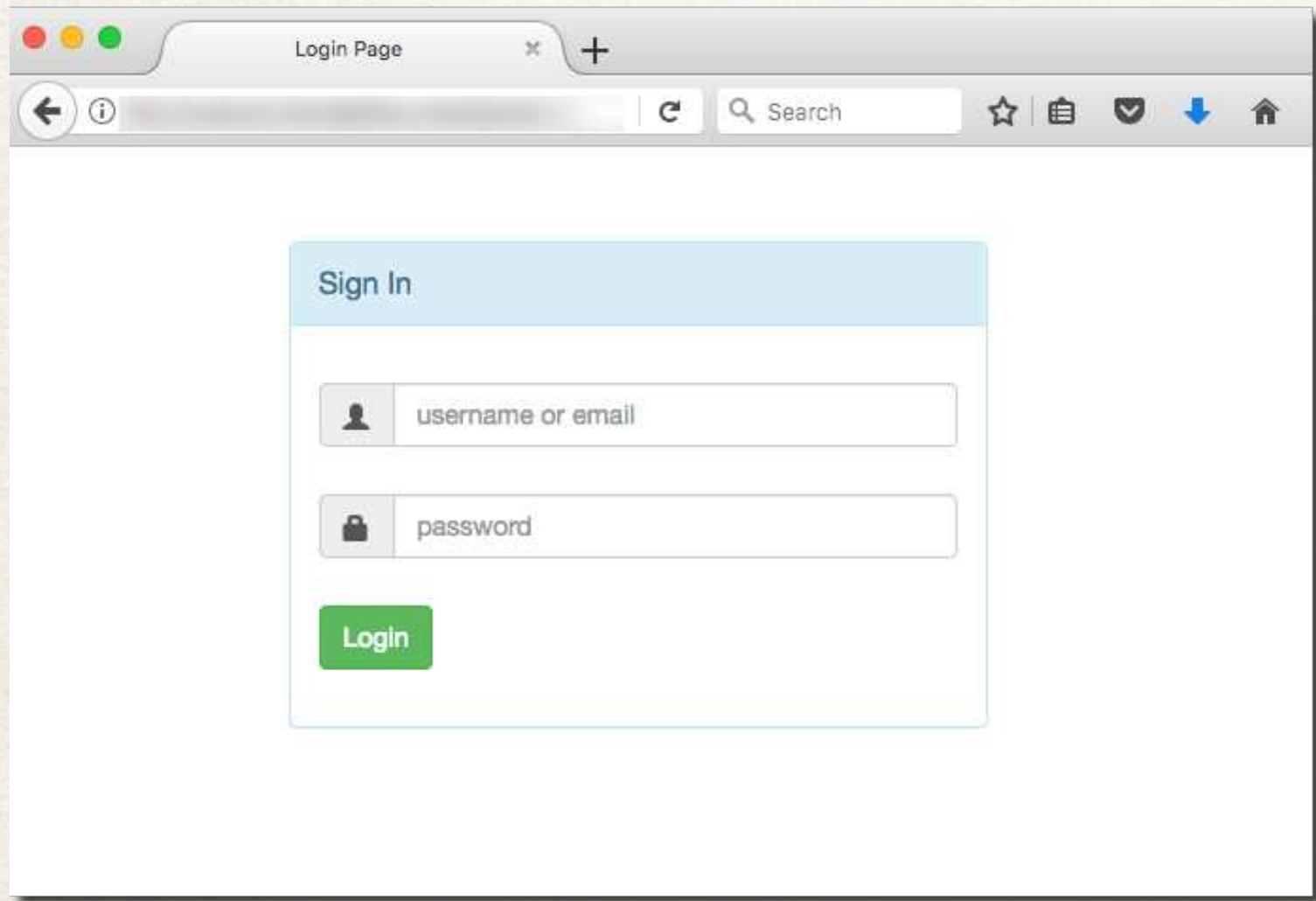


User
Roles

Restrict Access Based on Roles



Our Example

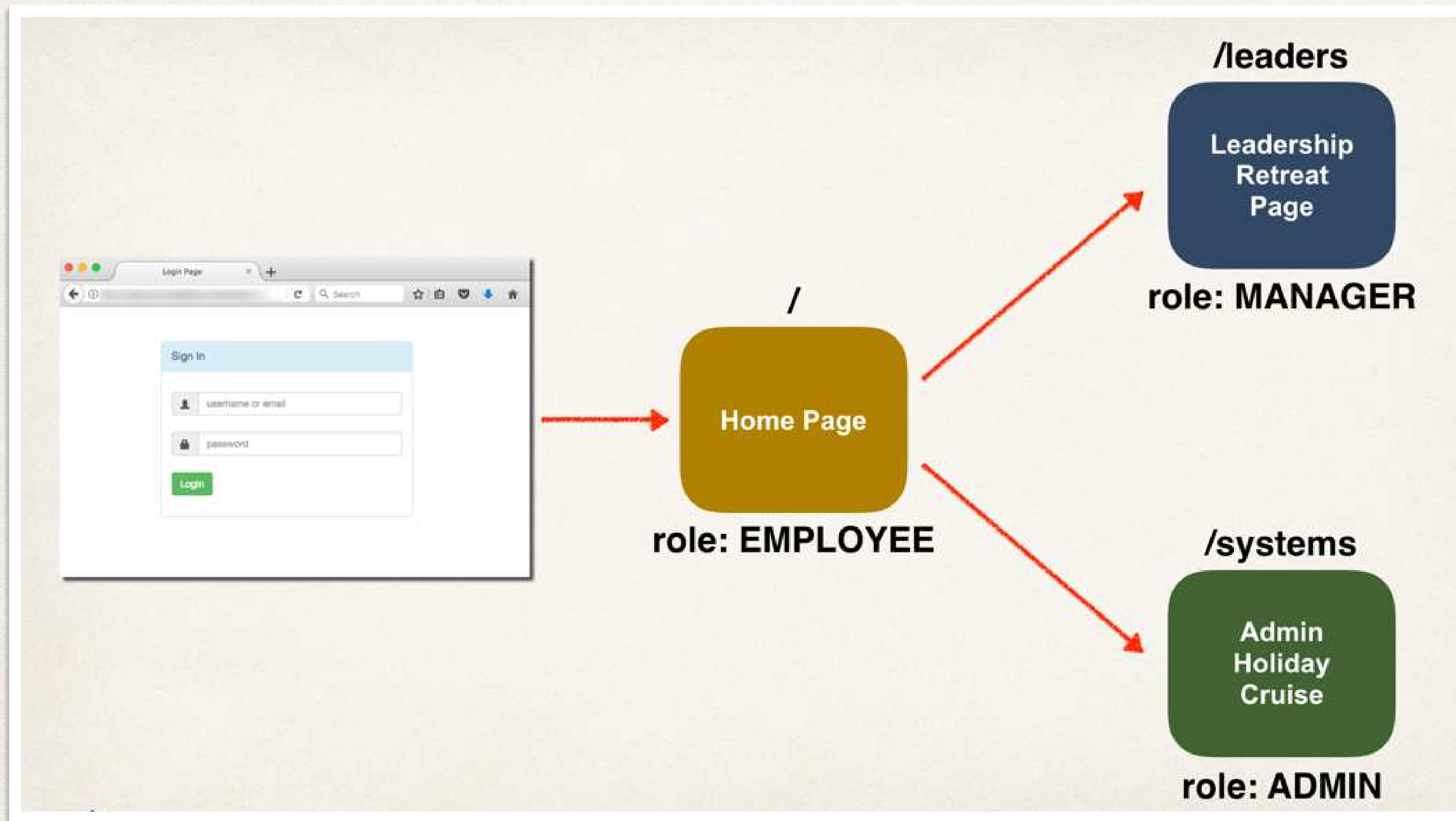


Development Process

Step-By-Step

1. Create supporting controller code and view pages
2. Restrict Access based on Roles

Step 1: Create Supporting Controller code and View Pages



Cover these steps
in the video

Step 2: Restricting Access to Roles

- Update your Spring Security Java configuration file (.java)
- General Syntax

Single role

```
requestMatchers(<< add path to match on >>).hasRole(<< authorized role >>)
```

Restrict access to
a given path
“/systems/**”

“ADMIN”

Step 3: Restricting Access to Roles

Any role in the list, comma-delimited list

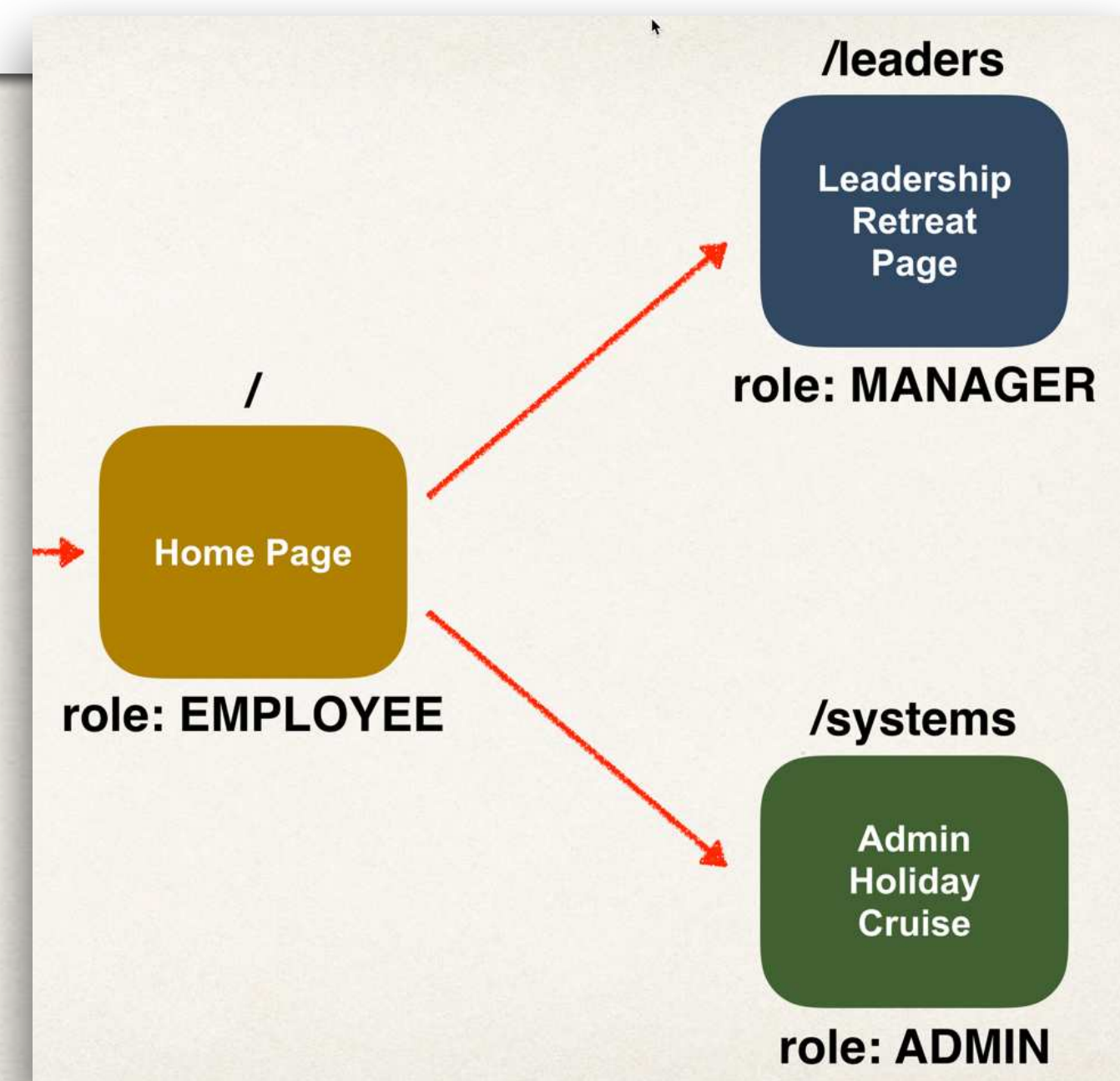
```
requestMatchers(<< add path to match on >>).hasAnyRole(<< list of authorized roles >>)
```

“ADMIN”, “DEVELOPER”, “VIP”, “PLATINUM”

Restrict Path to EMPLOYEE

```
requestMatchers("/").hasRole("EMPLOYEE")
```

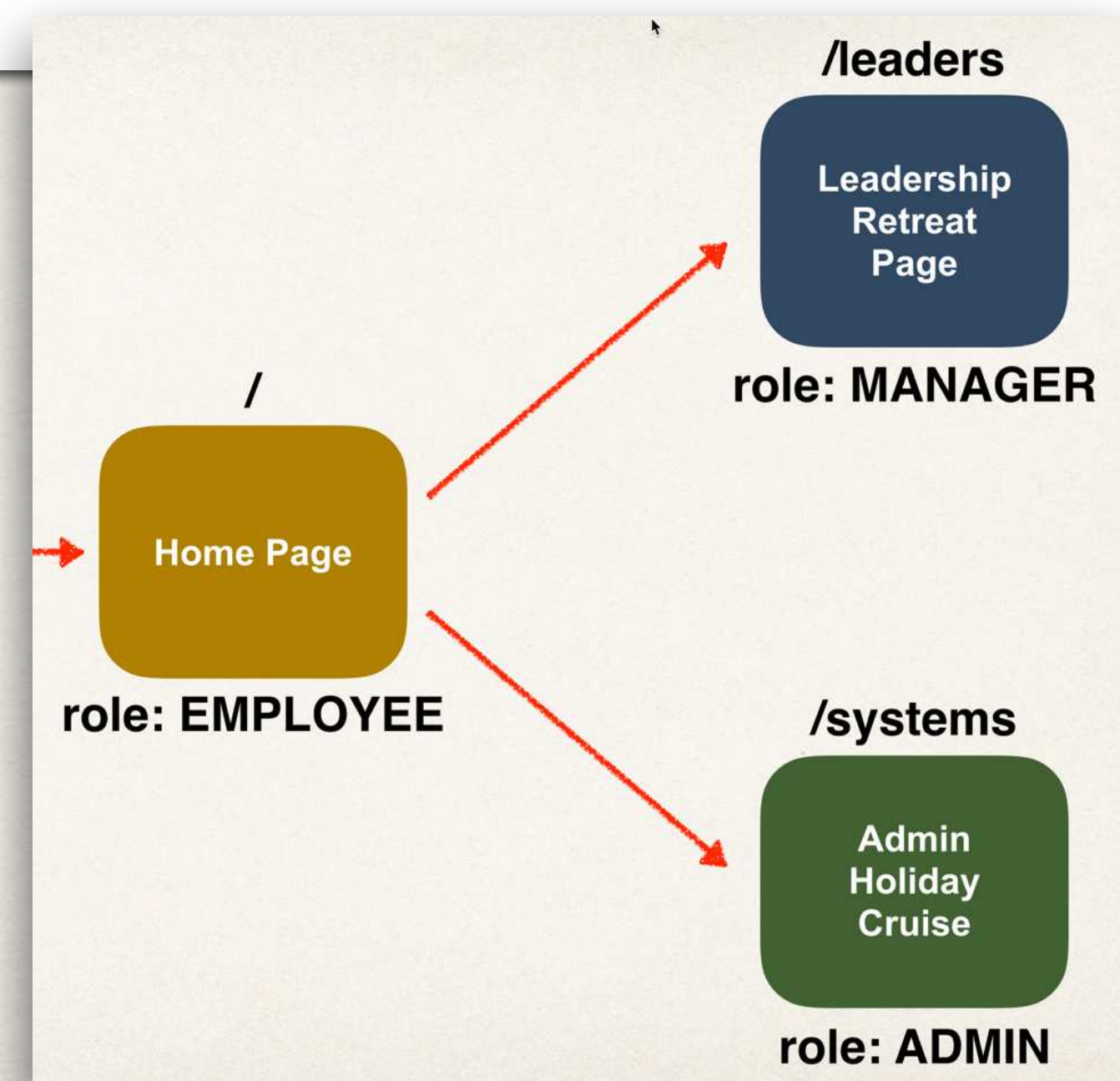
Match on root path: /



Restrict Path /leaders to MANAGER

```
requestMatchers("/leaders/**").hasRole("MANAGER")
```

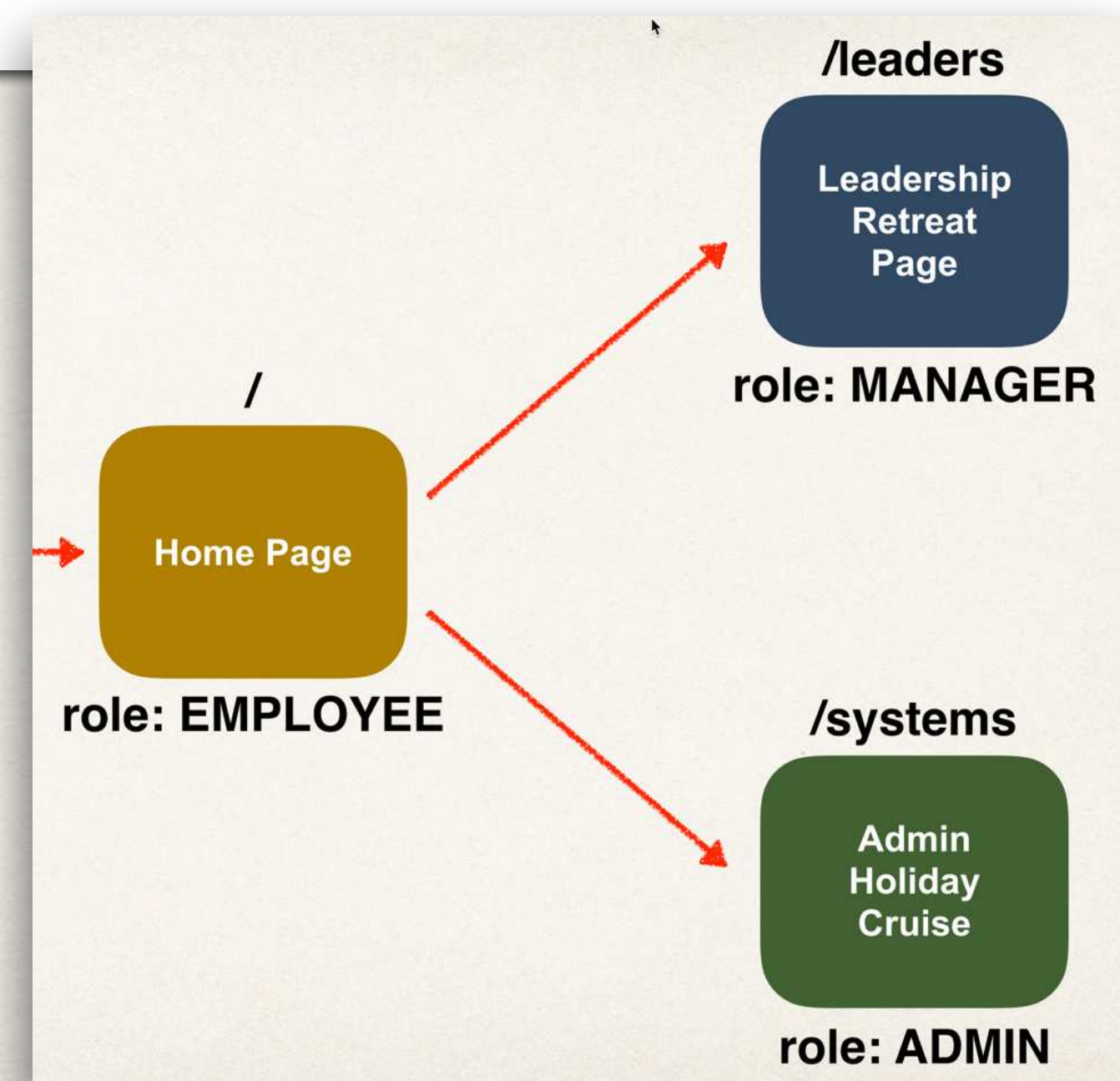
Match on path: /leaders
And all sub-directories (**)



Restrict Path `/systems` to ADMIN

```
requestMatchers("/systems/**").hasRole("ADMIN")
```

Match on path: `/systems`
And all sub-directories (`**`)

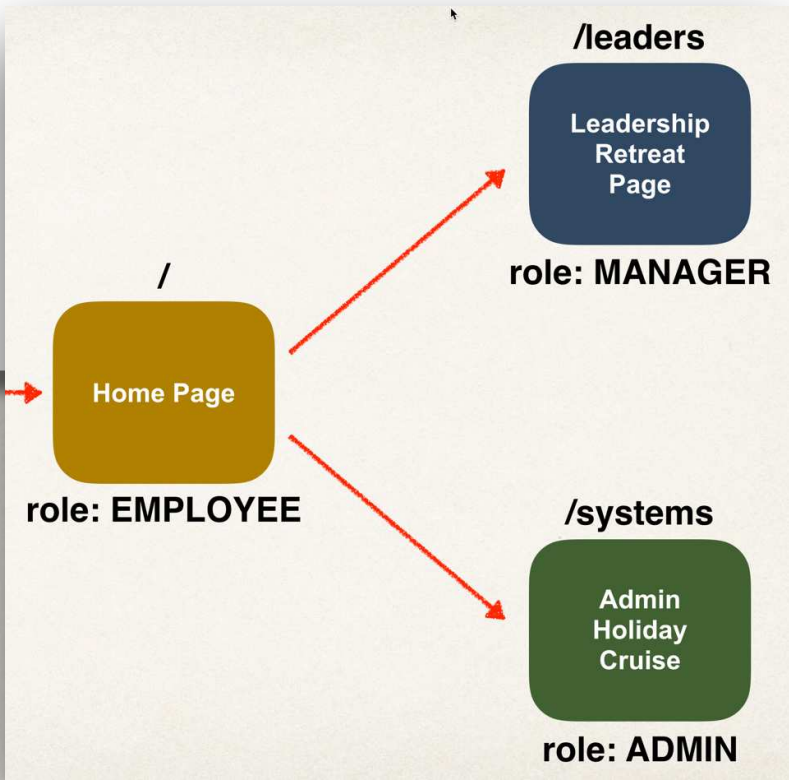


Pull It Together

@Bean

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
  
    http.authorizeHttpRequests(configurer ->  
        configurer  
            .requestMatchers("/").hasRole("EMPLOYEE")  
            .requestMatchers("/leaders/**").hasRole("MANAGER")  
            .requestMatchers("/systems/**").hasRole("ADMIN")  
            .anyRequest().authenticated()  
        )  
    )  
    ...  
}
```

User ID	Password	Roles
john	test123	EMPLOYEE
mary	test123	EMPLOYEE, MANAGER
susan	test123	EMPLOYEE, MANAGER, ADMIN



Custom Access Denied Page



Default Access Denied Page

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

There was an unexpected error (type=Forbidden, status=403).
Forbidden

Custom Access Denied Page

Access Denied - You are not authorized to access this resource.

[Back to Home Page](#)

You can customize this page

HTML + CSS

Development Process

Step-By-Step

1. Configure custom page for access denied
2. Create supporting controller code and view page

Step 1: Configure custom page access denied

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(configurer ->
        configurer
            .requestMatchers("/").hasRole("EMPLOYEE")
            ...
        )
        .exceptionHandling(configurer ->
            configurer
                .accessDeniedPage("/access-denied")
        );
    ...
}
```

Our request mapping path

Step 2: Create Supporting Controller code and View Page

Cover these steps
in the video

Display Content Based on Roles



Why Show These Links?

User: john

Role(s): [ROLE_EMPLOYEE]

[Leadership Meeting](#) (Only for Manager peeps)

[IT Systems Meeting](#) (Only for Admin peeps)

Logout

Since John is an employee,
he shouldn't be able to see this content / links

Display Content Based on Roles

Logged in
as Mary

User: mary

Role(s): [ROLE_EMPLOYEE, ROLE_MANAGER]

Leadership Meeting (Only for Manager peeps)

Logout

Mary can see
MANAGER content

Spring Security

Only show this section for users
with MANAGER role

```
...  
<div sec:authorize="hasRole('MANAGER')">
```

```
  <p>  
    <a th:href="@{/leaders}">  
      Leadership Meeting  
    </a>  
    (Only for Manager peeps)  
  </p>
```

```
</div>
```

User: mary

Role(s): [ROLE_EMPLOYEE, ROLE_MANAGER]

Leadership Meeting (Only for Manager peeps)

Display Content Based on Roles

Logged in
as Susan

User: susan

Role(s): [ROLE_ADMIN, ROLE_EMPLOYEE]

IT Systems Meeting (Only for Admin peeps)

Susan can see
ADMIN content

Spring Security

Only show this section for users
with ADMIN role

```
...  
<div sec:authorize="hasRole('ADMIN')">
```

```
  <p>  
    <a th:href="@{/systems}">  
      IT Systems Meeting  
    </a>  
    (Only for Admin peeps)  
  </p>
```

```
</div>
```

User: susan

Role(s): [ROLE_ADMIN, ROLE_EMPLOYEE]

[IT Systems Meeting](#) (Only for Admin peeps)

Spring Security

User Accounts Stored in Database



Database Access

- So far, our user accounts were hard coded in Java source code
- We want to add database access

Advanced

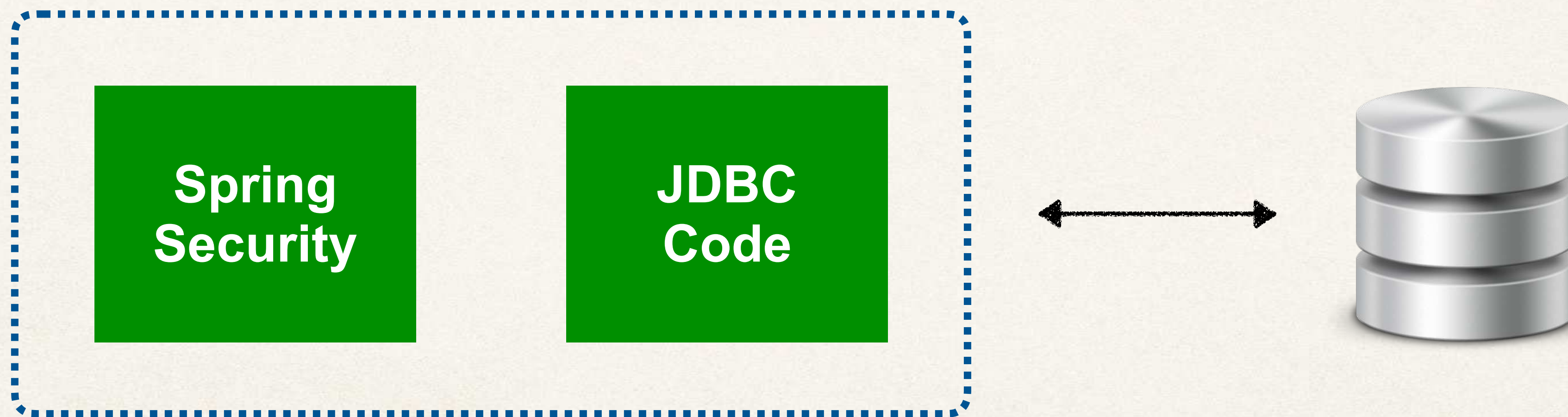
Recall Our User Roles

User ID	Password	Roles
john	test123	EMPLOYEE
mary	test123	EMPLOYEE, MANAGER
susan	test123	EMPLOYEE, MANAGER, ADMIN

Database Support in Spring Security

Out-of-the-box

- Spring Security can read user account info from database
- By default, you have to follow Spring Security's predefined table schemas



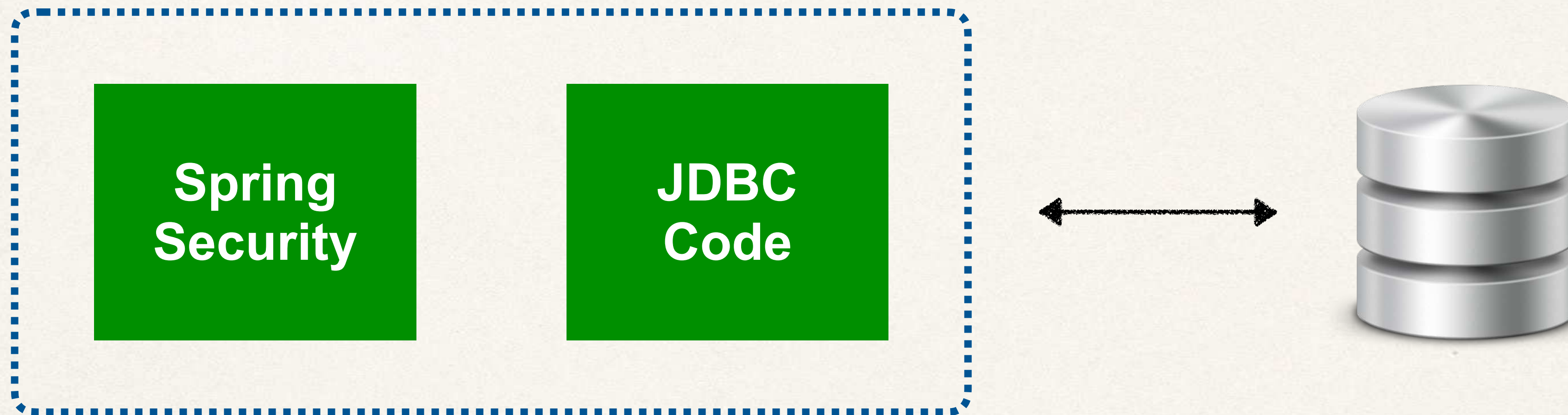
Customize Database Access with Spring Security

- Can also customize the table schemas
- Useful if you have custom tables specific to your project / custom
- You will be responsible for developing the code to access the data
 - JDBC, JPA / Hibernate etc ...

Database Support in Spring Security

Out-of-the-box

- Follow Spring Security's predefined table schemas



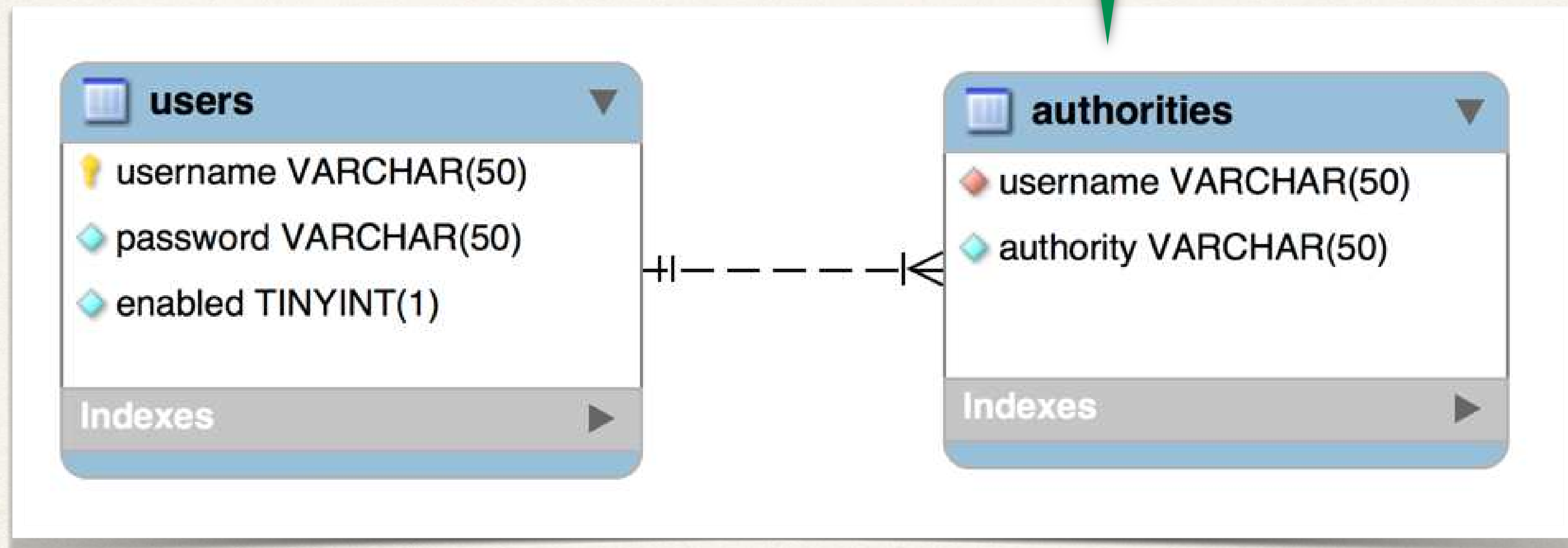
Development Process

Step-By-Step

1. Develop SQL Script to set up database tables
2. Add database support to Maven POM file
3. Create JDBC properties file
4. Update Spring Security Configuration to use JDBC

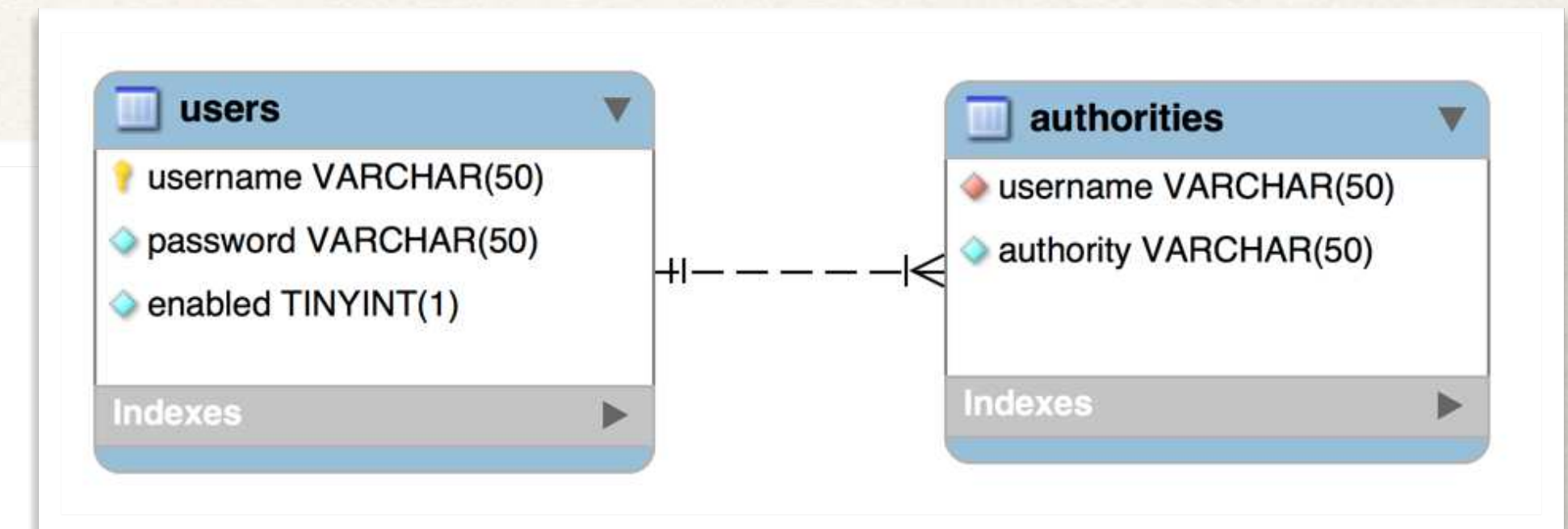
Default Spring Security Database Schema

“authorities” same as “roles”



Step 1: Develop SQL Script to setup database tables

```
CREATE TABLE `users` (  
  `username` varchar(50) NOT NULL,  
  `password` varchar(50) NOT NULL,  
  `enabled` tinyint NOT NULL,  
  
  PRIMARY KEY (`username`)  
  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```



Step 1: Develop SQL Script to setup database tables

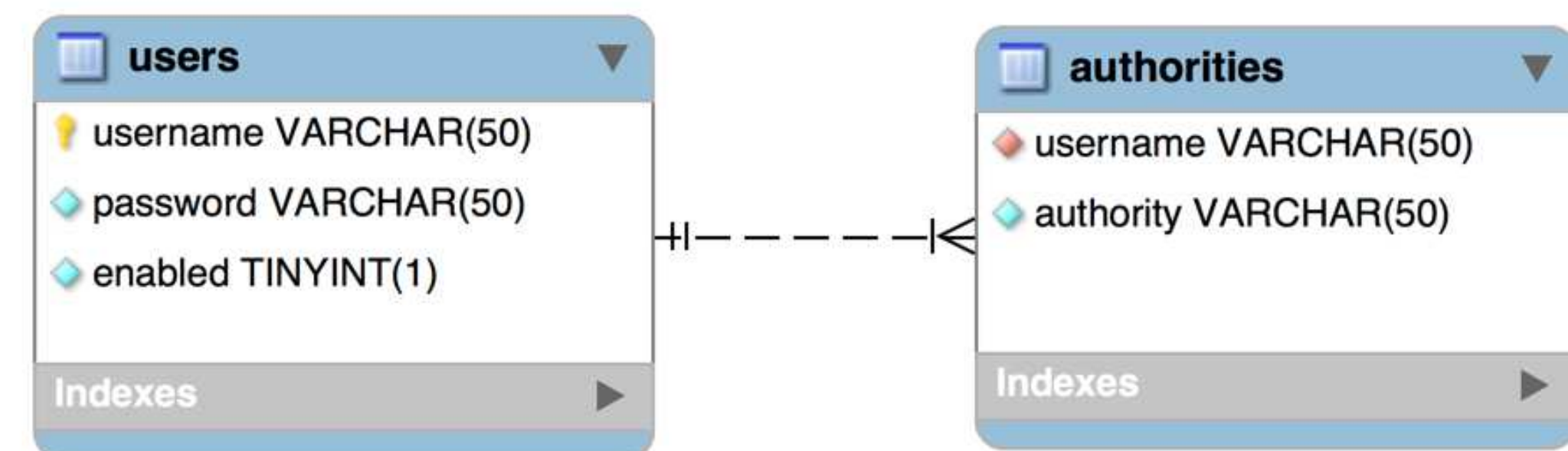
The encoding algorithm id

The password

```
INSERT INTO `users`  
VALUES  
( 'john' , ' {noop}test123' , 1) ,  
( 'mary' , ' {noop}test123' , 1) ,  
( 'susan' , ' {noop}test123' , 1) ;
```

User ID	Password	Roles
john	test123	EMPLOYEE
mary	test123	EMPLOYEE, MANAGER
susan	test123	EMPLOYEE, MANAGER, ADMIN

Let's Spring Security know the passwords are stored as plain text (noop)



Step 1: Develop SQL Script to setup database tables

```
CREATE TABLE `authorities` (  
  `username` varchar(50) NOT NULL,  
  `authority` varchar(50) NOT NULL,
```

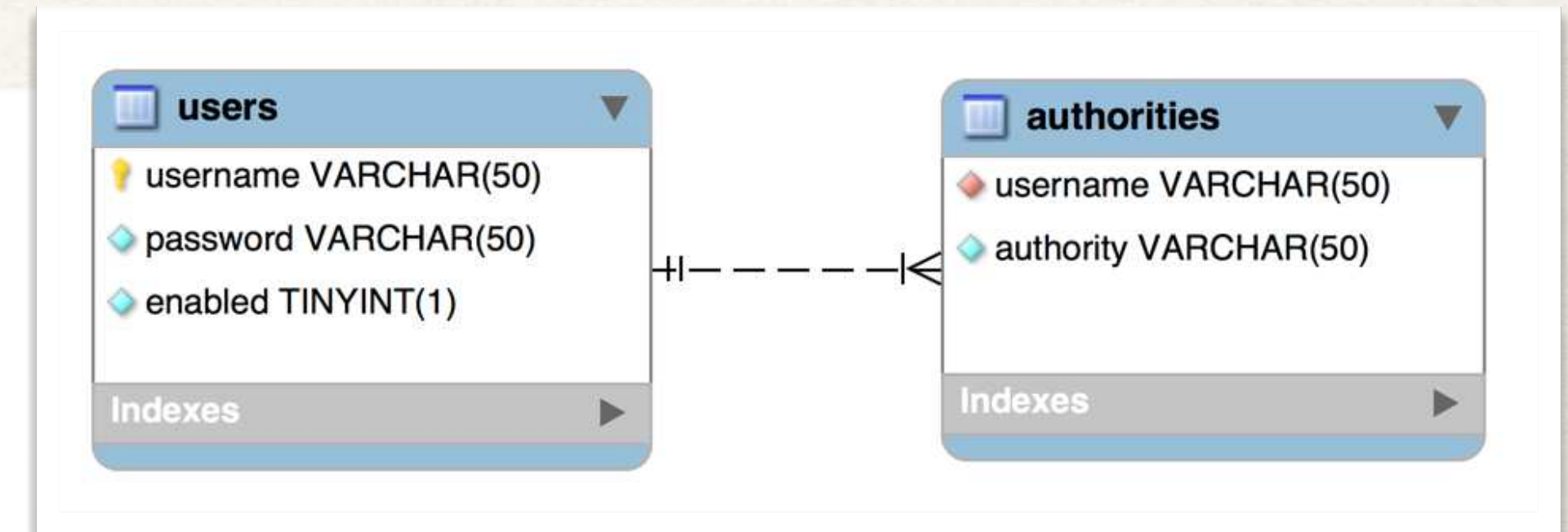
```
  UNIQUE KEY `authorities_idx_1` (`username`,`authority`),
```

```
  CONSTRAINT `authorities_ibfk_1`
```

```
  FOREIGN KEY (`username`)
```

```
  REFERENCES `users` (`username`)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```



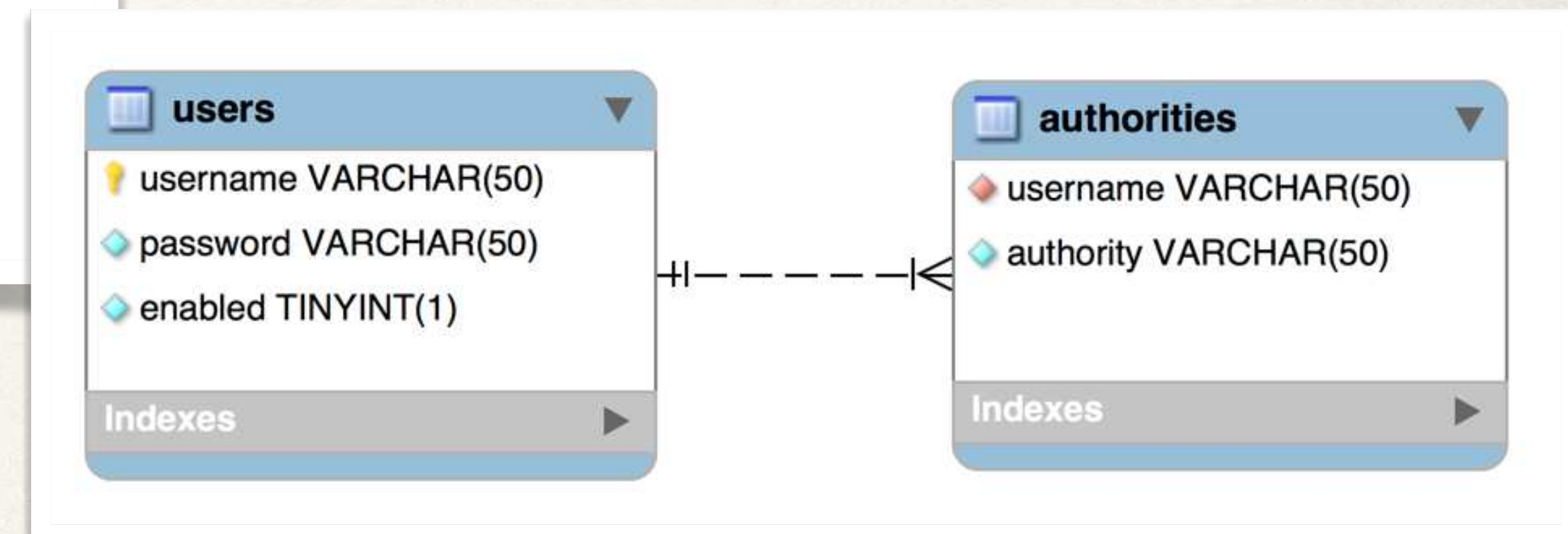
Step 1: Develop SQL Script to setup database tables

“authorities” same as “roles”

```
INSERT INTO `authorities`  
VALUES  
( 'john', 'ROLE_EMPLOYEE' ),  
( 'mary', 'ROLE_EMPLOYEE' ),  
( 'mary', 'ROLE_MANAGER' ),  
( 'susan', 'ROLE_EMPLOYEE' ),  
( 'susan', 'ROLE_MANAGER' ),  
( 'susan', 'ROLE_ADMIN' );
```

Internally Spring Security uses
“ROLE_” prefix

User ID	Password	Roles
john	test123	EMPLOYEE
mary	test123	EMPLOYEE, MANAGER
susan	test123	EMPLOYEE, MANAGER, ADMIN



Step 2: Add Database Support to Maven POM file

```
<!-- MySQL -->  
<dependency>  
  <groupId>com.mysql</groupId>  
  <artifactId>mysql-connector-j</artifactId>  
  <scope>runtime</scope>  
</dependency>
```

JDBC Driver

Step 3: Create JDBC Properties File

File: application.properties

```
#  
# JDBC connection properties  
#  
spring.datasource.url=jdbc:mysql://localhost:3306/employee_directory  
spring.datasource.username=springstudent  
spring.datasource.password=springstudent
```


Step 4: Update Spring Security to use JDBC

Inject data source
Auto-configured by Spring Boot

```
@Configuration
public class DemoSecurityConfig {

    @Bean
    public UserDetailsManager userDetailsManager(DataSource dataSource) {

        return new JdbcUserDetailsManager(dataSource);
    }

    ...
}
```

No longer
hard-coding users :-)

Tell Spring Security to use
JDBC authentication
with our data source

Spring Security Password Encryption



Password Storage

- So far, our user passwords are stored in plaintext ... yikes!

username	password	enabled
john	{noop}test123	1
mary	{noop}test123	1
susan	{noop}test123	1

- Ok for getting started ... but not for production / real-time project :-)

Password Storage - Best Practice

Best Practice

- The best practice is store passwords in an encrypted format

username	password	enabled
john	{bcrypt}\$2a\$10\$qeS0HEh7urweMojsnwNAR.vcXJeXR1UcMRZ2WcGQl9YeuspUdgF.q	1
mary	{bcrypt}\$2a\$10\$qeS0HEh7urweMojsnwNAR.vcXJeXR1UcMRZ2WcGQl9YeuspUdgF.q	1
susan	{bcrypt}\$2a\$10\$qeS0HEh7urweMojsnwNAR.vcXJeXR1UcMRZ2WcGQl9YeuspUdgF.q	1

Encrypted version of password

Spring Security Team Recommendation

- Spring Security recommends using the popular **bcrypt** algorithm
- bcrypt
 - Performs one-way encrypted hashing
 - Adds a random salt to the password for additional protection
 - Includes support to defeat brute force attacks

Bcrypt Additional Information

- Why you should use bcrypt to hash passwords

www.luv2code.com/why-bcrypt

- Detailed bcrypt algorithm analysis

www.luv2code.com/bcrypt-wiki-page

- Password hashing - Best Practices

www.luv2code.com/password-hashing-best-practices

How to Get a Bcrypt password

You have a plaintext password and you want to encrypt using bcrypt

- Option 1: Use a website utility to perform the encryption
- Option 2: Write Java code to perform the encryption

How to Get a Bcrypt password - Website

- Visit: **`www.luv2code.com/generate-bcrypt-password`**
- Enter your plaintext password
- The website will generate a bcrypt password for you

DEMO

Development Process

Step-By-Step

1. Run SQL Script that contains encrypted passwords

- Modify DDL for password field, length should be 68

THAT'S IT ... no need to change Java source code :-)

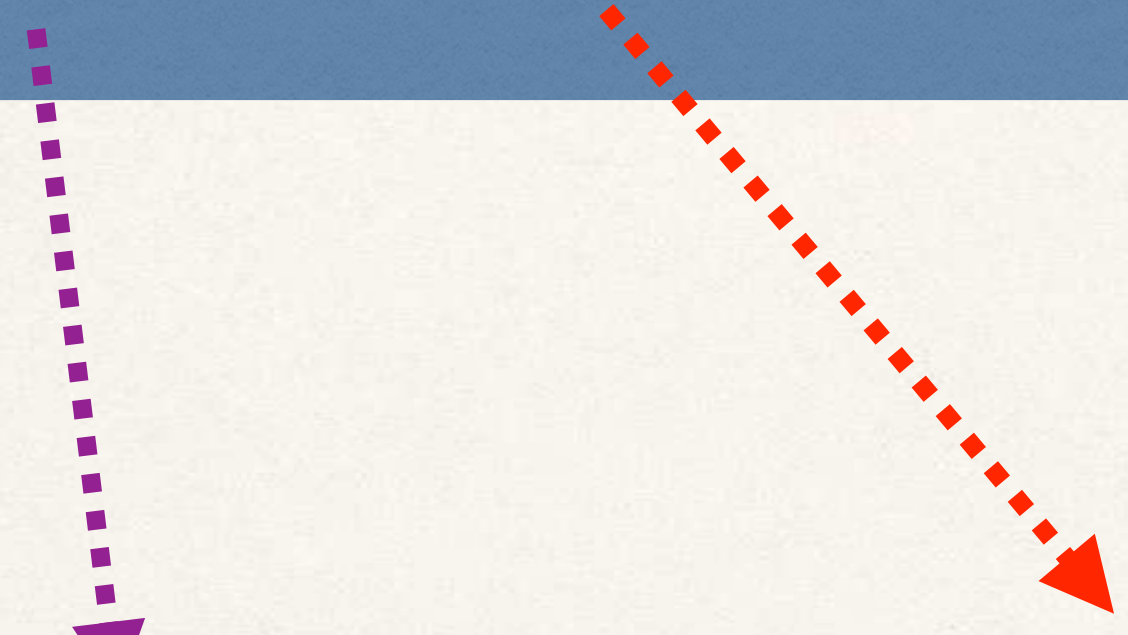
Spring Security Password Storage

- In Spring Security, passwords are stored using a specific format

`{bcrypt}encodedPassword`

Password column must be at least 68 chars wide

`{bcrypt}` - 8 chars
encodedPassword - 60 chars



username	password	enabled
john	{bcrypt}\$2a\$10\$qeS0HEh7urweMojsnwNAR.vcXJeXR1UcMRZ2WcGQl9YeuspUdgF.q	1
mary	{bcrypt}\$2a\$10\$qeS0HEh7urweMojsnwNAR.vcXJeXR1UcMRZ2WcGQl9YeuspUdgF.q	1
susan	{bcrypt}\$2a\$10\$qeS0HEh7urweMojsnwNAR.vcXJeXR1UcMRZ2WcGQl9YeuspUdgF.q	1

Modify DDL for Password Field

```
CREATE TABLE `users` (  
  `username` varchar(50) NOT NULL,  
  password char(68) NOT NULL,  
  `enabled` tinyint NOT NULL,  
  
  PRIMARY KEY (`username`)  
  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Password column must be at least 68 chars wide

{bcrypt} - 8 chars

encodedPassword - 60 chars

Step 1: Develop SQL Script to setup database tables

The encoding
algorithm id

The encrypted password: fun123

```
INSERT INTO `users`  
VALUES  
( 'john' , '{bcrypt}$2a$10$qeS0HEh7urweMojsnwNAR.vcXJeXR1UcMRZ2WcGQ19YeuspUdgF.q' , 1) ,  
...
```

Let's Spring Security know the
passwords are stored as
encrypted passwords: bcrypt

Step 1: Develop SQL Script to setup database tables

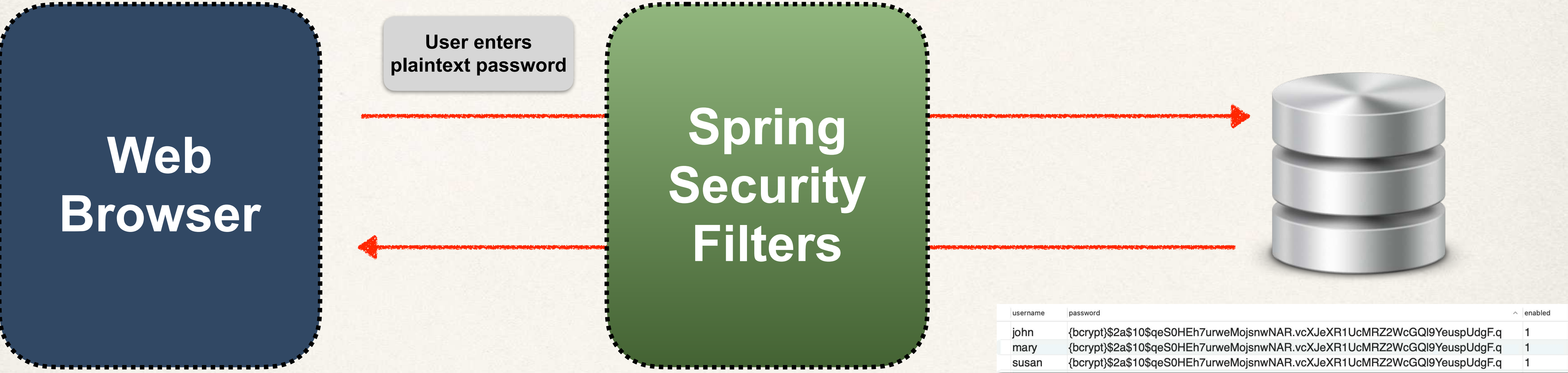
The encoding
algorithm id

The encrypted password: fun123

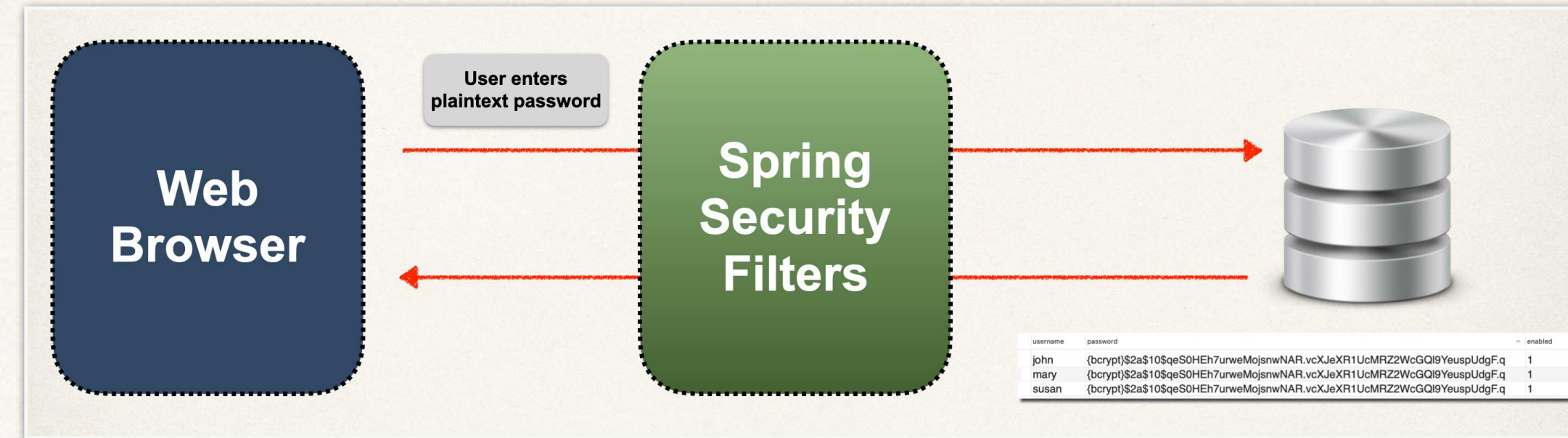
```
INSERT INTO `users`  
VALUES
```

```
( 'john' , '{bcrypt}$2a$10$qeS0HEh7urweMojsnwNAR.vcXJeXR1UcMRZ2WcGQ19YeuspUdgF.q' , 1) ,  
( 'mary' , '{bcrypt}$2a$04$eFytJDGtjbThXa80FyOOBuFdK2IwjyWefYkMpiBEFlpBwDH.5PM0K' , 1) ,  
( 'susan' , '{bcrypt}$2a$04$eFytJDGtjbThXa80FyOOBuFdK2IwjyWefYkMpiBEFlpBwDH.5PM0K' , 1) ;
```


Spring Security Login Process



Spring Security Login Process



Note:

The password from db is
NEVER decrypted

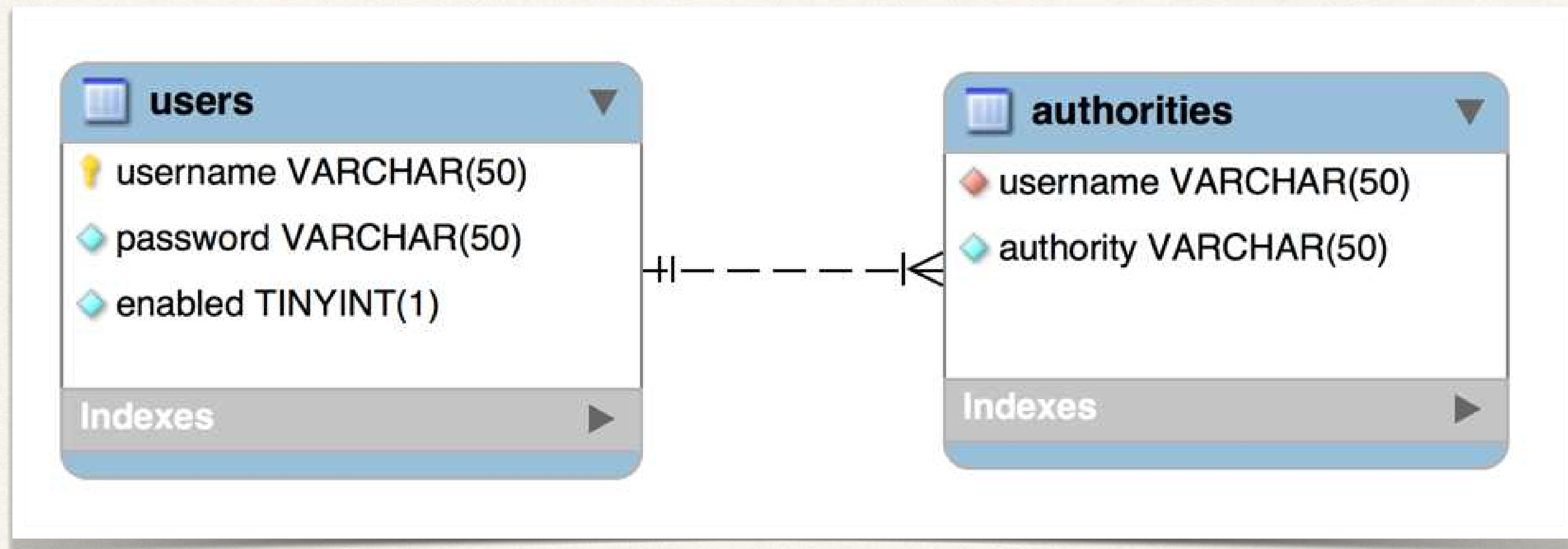
Because bcrypt is a
one-way
encryption algorithm

1. Retrieve password from db for the user
2. Read the encoding algorithm id (bcrypt etc)
3. For case of bcrypt, encrypt plaintext password from login form (using salt from db password)
4. Compare encrypted password from login form WITH encrypted password from db
5. If there is a match, login successful
6. If no match, login NOT successful

Spring Security Custom Tables

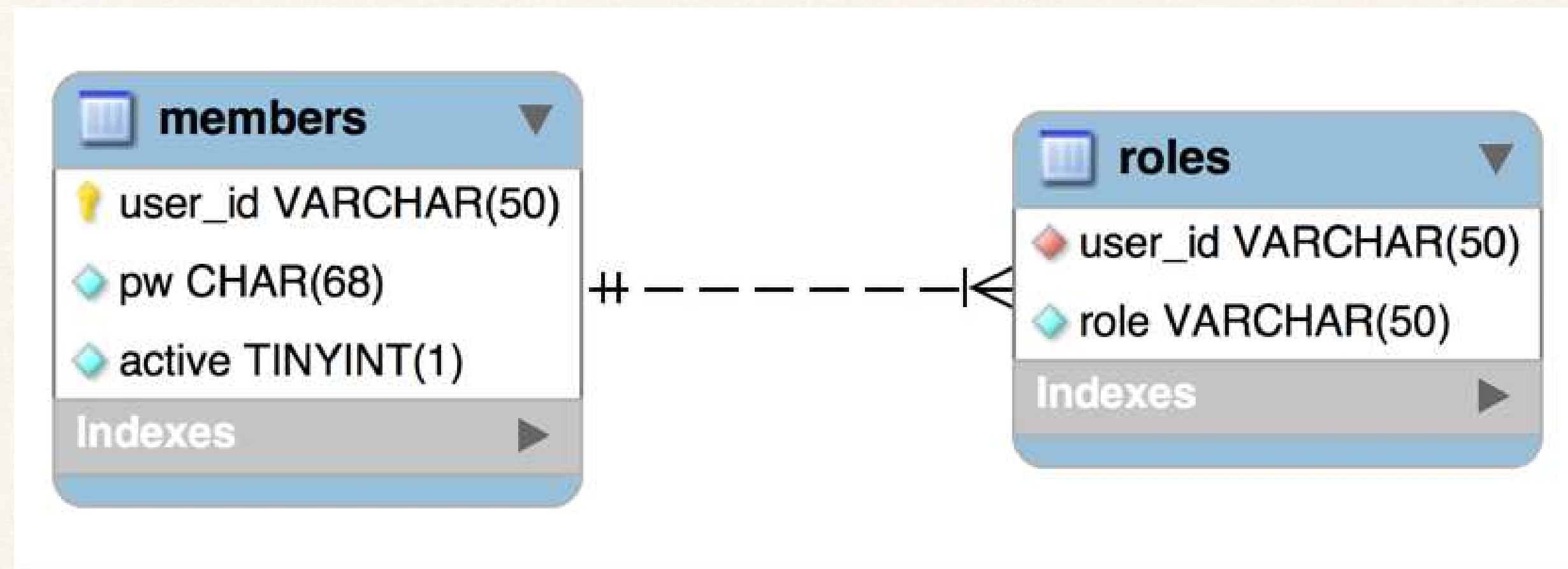


Default Spring Security Database Schema



Custom Tables

- What if we have our own custom tables?
- Our own custom column names?



This is all custom
Nothing matches with default Spring Security table schema

For Security Schema Customization

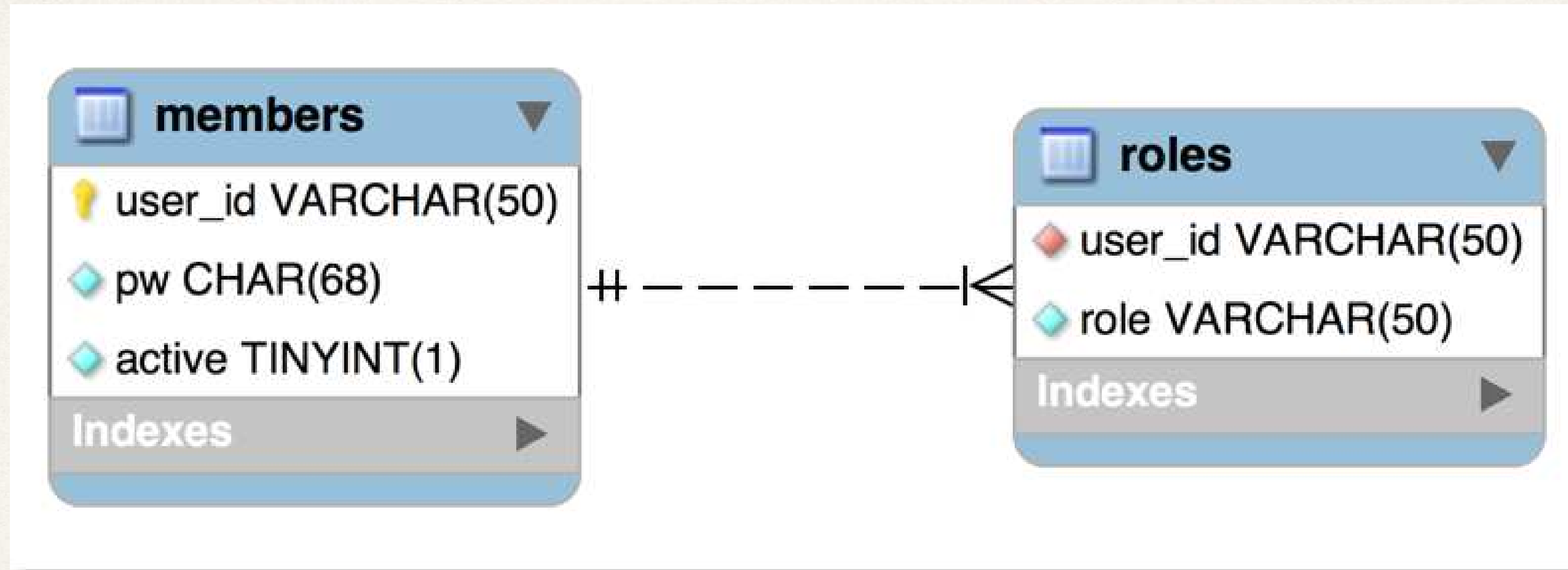
- Tell Spring how to query your custom tables
- Provide query to find user by user name
- Provide query to find authorities / roles by user name

Development Process

Step-By-Step

1. Create our custom tables with SQL
2. Update Spring Security Configuration
 - Provide query to find user by user name
 - Provide query to find authorities / roles by user name

Step 1: Create our custom tables with SQL



This is all custom
Nothing matches with default Spring Security table schema

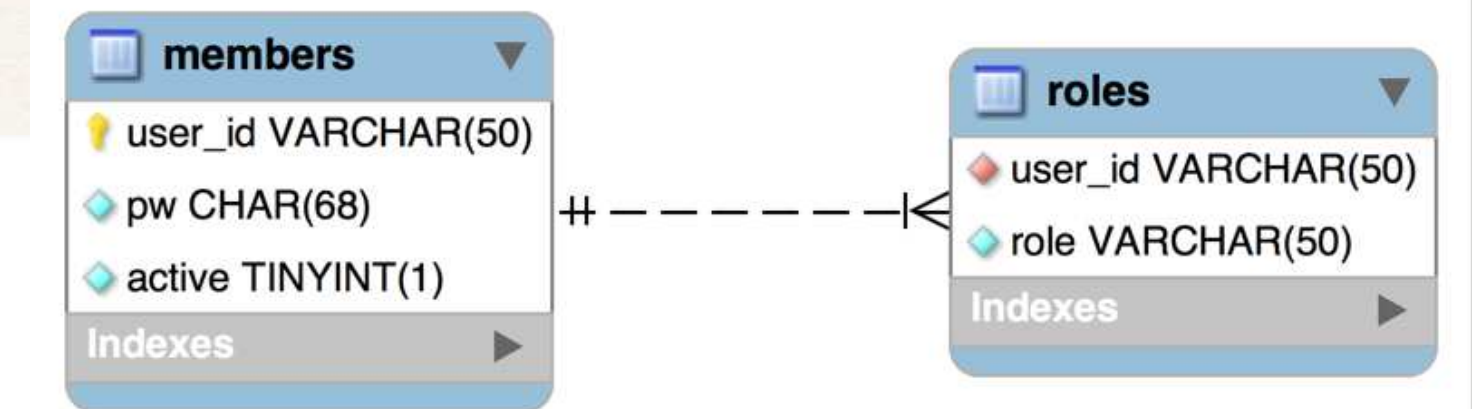
Step 2: Update Spring Security Configuration

@Configuration

```
public class DemoSecurityConfig {
```

@Bean

```
public UserDetailsManager userDetailsManager(DataSource dataSource) {  
    JdbcUserDetailsManager theUserDetailsManager = new JdbcUserDetailsManager(dataSource);  
  
    theUserDetailsManager  
        .setUsersByUsernameQuery("select user_id, pw, active from members where user_id=?");  
  
    theUserDetailsManager  
        .setAuthoritiesByUsernameQuery("select user_id, role from roles where user_id=?");  
  
    return theUserDetailsManager;  
}  
...  
}
```



Step 2: Update Spring Security Configuration

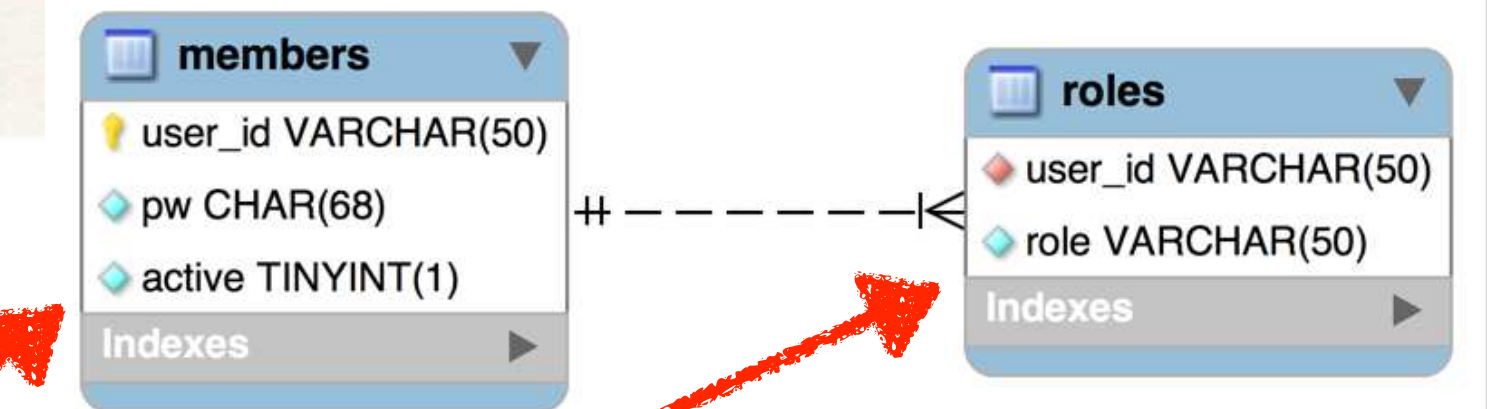
```
@Configuration
public class DemoSecurityConfig {

    @Bean
    public UserDetailsManager userDetailsManager(DataSource dataSource) {
        JdbcUserDetailsManager theUserDetailsManager = new JdbcUserDetailsManager(dataSource);

        theUserDetailsManager
            .setUsersByUsernameQuery("select user_id, pw, active from members where user_id=?");

        theUserDetailsManager
            .setAuthoritiesByUsernameQuery("select user_id, role from roles where user_id=?");

        return theUserDetailsManager;
    }
    ...
}
```



How to find users

How to find roles

Question mark "?"
Parameter value will be the
user name from login